StarPU Handbook

for StarPU 1.4.5

Generated by Doxygen.

VIII Appendix 317

This manual documents the usage of StarPU version 1.4.5. Its contents was last updated on 2024-04-19.

Chapter 1

Introduction

1.1 we need to keep 2 blank lines above

1.2 Motivation

The use of specialized hardware, such as accelerators or coprocessors offers an interesting approach to overcoming the physical limits encountered by processor architects. As a result, many machines are now equipped with one or several accelerators (e.g. a GPU), in addition to the usual processor(s). While significant efforts have been devoted to offloading computation onto such accelerators, very little attention has been paid to portability concerns on the one hand, and to the possibility of having heterogeneous accelerators and processors interact on the other hand.

StarPU is a runtime system that provides support for heterogeneous multicore architectures. It not only offers a unified view of the computational resources (i.e. CPUs and accelerators simultaneously) but also takes care of efficiently mapping and executing tasks onto an heterogeneous machine while transparently handling low-level issues such as data transfers in a portable manner.

1.3 StarPU in a Nutshell

StarPU is a software tool designed to enable programmers to harness the computational capabilities of both CPUs and GPUs, all while sparing them the need to meticulously adapt their programs for specific target machines and processing units.

At the heart of StarPU lies its runtime support library, which takes charge of scheduling tasks supplied by applications on heterogeneous CPU/GPU systems. Furthermore, StarPU provides programming language support through an OpenCL front-end (SOCL OpenCL Extensions).

StarPU's runtime mechanism and programming language extensions are built around a task-based programming model. In this modell, applications submit computational tasks, with CPU and/or GPU implementations. StarPU effectively schedules these tasks and manages the associated data transfers across available CPUs and GPUs. The data that a task operates on are automatically exchanged between accelerators and the main memory, thereby sparing programmers the intricacies of scheduling and the technical details tied to these transfers.

StarPU excels in its adaptness at efficiently scheduling tasks using established algorithms from the literature (Task Scheduling Policies). Furthermore addition, it provides the flexibility for scheduling experts, such as compiler or computational library developers, to implement custom scheduling policies in a manner that is easily portable (How To Define A New Scheduling Policy).

The remainder of this section describes the main concepts used in StarPU.

A video, lasting 26 minutes, accessible on the StarPU website (`https://starpu.gitlabpages.inria.fr/`) presents these concepts.

Additionally, a serie of tutorials can be found at `https://starpu.gitlabpages.inria.fr/tutorials/`

1.3.1 Codelet and Tasks

One of StarPU's key data structures is the **codelet**. A codelet defines a computational kernel that can potentially be implemented across various architectures, including CPUs, CUDA devices, or OpenCL devices.

Another pivotal data structure is the **task**. Executing a StarPU task involves applying a codelet to a data set,

utilizing one of the architectures on which the codelet is implemented. Therefore, a task describes the codelet that it uses, the data accessed, and how they are accessed during the computation (read and/or write). StarPU tasks are asynchronous, meaning that submitting a task to StarPU is a non-blocking operation. The task structure can also specify a **callback** function, which is called once StarPU succesfully completes the task. Additionally, it contains optional fields that the application may use to provide hints to the scheduler, such as priority levels.
By default, task dependencies are inferred from data dependency (sequential coherency) within StarPU. However, the application has the ability to disable sequential coherency for specific data, and dependencies can also be specifically defined. A task can be uniquely identified by a 64-bit number, chosen by the application, referred to as a **tag**. Task dependencies can be enforced through callback functions, by submitting other tasks, or by specifying dependencies between tags (which can correspond to tasks that have yet to be submitted).

1.3.2 StarPU Data Management Library

As StarPU dynamically schedules tasks at runtime, the need for data transfers is automatically managed in a`‘just-in-time'' manner between different processing units, This automated approach alleviates the burden on application programmers to explicitly handle data transfers. Furthemore, to minimize needless transfers, StarPU retains data at the location of its last use, even if modifications were made there. Additionally, StarPU allows multiple instances of the same data to coexist across various processing units simultaneously, as long as the data remains unaltered.

1.4 Application Taskification

We will explain here shortly the concept of "taskifying" an application.
Before transitioning to StarPU, you must transform your application as follows:

- Refactor functions into "pure" functions that exclusively utilize data from their parameters.
- Create a central main function responsible for calling these pure functions.

Once this restructuring is complete, integrating StarPU or any similar task-based library becomes straightforward. You merely replace function calls with task submissions, leveraging the library's capabilities.
Chapter A Stencil Application shows how to easily convert an existing application to use StarPU.

1.5 Research Papers

Research papers about StarPU can be found at `https://starpu.gitlabpages.inria.fr/publications/`.
A good overview is available in the research report at `http://hal.archives-ouvertes.fr/inria-00467677`.

Chapter 2

Documentation Organization

The documentation chapters include

- **------— StarPU Installation ------—**
    - Building and Installing StarPU
    - Execution Configuration Through Environment Variables
    - Compilation Configuration
- **------— StarPU Basics ------—**
    - StarPU Applications, setting up Your Own Code
    - Basic Examples
    - Full source code for the ’Scaling a Vector’ example
    - Tasks In StarPU
    - Data Management
    - Scheduling
    - Examples in StarPU Sources
- **------— StarPU Applications ------—**
    - A Stencil Application
- **------— StarPU Performances ------—**
    - Benchmarking StarPU
    - Online Performance Tools
    - Offline Performance Tools
- **------— StarPU FAQs ------—**
    - Check List When Performance Are Not There
    - Frequently Asked Questions
- **------— StarPU Language Bindings ------—**
    - The StarPU Native Fortran Support
    - StarPU Java Interface
    - Python Interface
    - The StarPU OpenMP Runtime Support (SORS)
- **------— StarPU Extensions ------—**
    - Configuration and Initialization
    - Advanced Tasks In StarPU

  - Advanced Data Management
  - Advanced Scheduling
  - Scheduling Contexts
  - Scheduling Context Hypervisor
  - How To Define A New Scheduling Policy
  - CUDA Support
  - OpenCL Support
  - Maxeler FPGA Support
  - Out Of Core
  - MPI Support
  - TCP/IP Support
  - Transactions
  - Fault Tolerance
  - FFT Support
  - SOCL OpenCL Extensions
  - Hierarchical DAGS
  - Creating Parallel Workers On A Machine
  - Interoperability Support
  - SimGrid Support
  - Debugging Tools
  - Helpers
- **Appendices**
  - The GNU Free Documentation License
  - Module Documentation
  - File Documentation
  - Deprecated List

Make sure to have had a look at those too!

Chapter 3

Glossary

A **codelet** stores pointers to different implementations of the same theoretical function.
A **memory node** can be either the main RAM, GPU-embedded memory or disk memory.
A **bus** represents a connection between memory nodes.
A **data handle** keeps track of multiple copies of the same data (**registered** by the application) across various memory nodes. The data management library ensures coherency among these copies.
The **home** memory node of a data handle is the memory node where the data was originally registered (typically the main memory node).
A **task** represents a scheduled execution of a codelet on specific data handles.
A **tag** is a rendez-vous point. Tasks generally have their own tag and can depend on other tags. The value of a tag is chosen by the application.
A **worker** execute tasks. Typically, there is one worker per CPU computation core and one per accelerator (with a dedicated whole CPU core).
A **driver** oversees a given type of worker. Currently, there are CPU, CUDA, and OpenCL drivers.
A **performance model** is a (dynamic or static) model of the performance of a given codelet. Codelets can have performance model for execution time as well as energy consumption.
A data **interface** describes the layout of the data: for a vector, it includes a pointer for the start, the number of elements and the size of elements ; for a matrix, it involves a pointer for the start, the number of elements per row, the offset between rows, and the size of each element ; etc. Codelet functions receive interfaces for the local memory node copies of data handles assigned to the scheduled task, to access their data.
Data **partitioning** means dividing the data of a specific data handle (referred to as the **father**) into several **children** data handles, each representing distinct segments of the original data.
A **filter** is the function responsible for deriving child data handles from a father data handle, thus defining how the partitioning should be done (e.g. horizontal, vertical, etc.)
Acquiring a data handle can be done from the main application, allowing secure access to the data of a data handle from its home node without needing to unregister it.

Part I

StarPU Installation

Chapter 4

Organization

This parts shows a basic usage of StarPU and how to execute the provided examples or your own applications.

- Chapter Building and Installing StarPU shows how to build and install StarPU.
- Chapter Compilation Configuration shows how to tune StarPU building process through configuration options.
- Chapter Execution Configuration Through Environment Variables lists environment variables that can be used to tune StarPU when executing an application.

Finally, Chapter Configuration and Initialization shows a brief overview of how to configure and tune StarPU.

Chapter 5

Building and Installing StarPU

Depending on the level of customization required for the library installation, we offer several solutions.

1. **Basic Installation or Evaluation:** If you are looking to simply try out the library, assess its performance on simple cases, run examples, or use the latest stable version, we recommend the following options:
    - For Linux Debian or Ubuntu distributions, consider using the latest StarPU Debian package (see Installing a Binary Package).
    - For macOS, you can opt for Brew and follow the steps in Installing a Source Package.
    - Using an already installed module on a cluster, as explained in Using a Module
2. **Customization for Specific Needs:** If you intend to use StarPU but require modifications, such as switching to another version (git branch), changing the default MPI, utilizing a preferred compiler, or altering source code, consider these options:
    - Guix or Spack can be useful, as these package managers allow dynamic changes during source-based builds. Refer to Installing a Source Package for details.
    - Alternatively, you can directly build from the source using the native build system of the library (Makefile, GNU autotools). Instructions can be found in Building from Source.
3. **Experiment Reproducibility:** If your focus is on experiment reproducibility, we recommend using Guix. Refer to Installing a Source Package for guidance.

Whichever solution you choose, you can utilize the tool `bin/starpu_config` to view all the configuration parameters used during StarPU installation.
Please refer to the provided documentation for specific installation steps and details for each solution.

5.1 Installing a Binary Package

One of the StarPU developers being a Debian Developer, the packages are well integrated and very up-to-date. To see which packages are available, simply type:

```
$ apt-cache search starpu
```

To install what you need, type for example:

```
$ sudo apt-get install libstarpu-dev
```

5.2 Installing a Source Package

StarPU is available from different package managers.

- Guix `https://gitlab.inria.fr/guix-hpc/guix-hpc`
- Spack `https://github.com/spack/spack/`
- Brew `https://gitlab.inria.fr/solverstack/brew-repo`

Documentation on how to install StarPU with these package managers is directly available from the links specified above. We give below a brief overview of the spack installation.

5.2.1 Installing the Spack Package

Here is a quick guide to install StarPU with spack.

```
$ git clone git@github.com:spack/spack.git
$ source ./spack/share/spack/setup-env.sh # if you use bash or zsh
$ spack install starpu
```

By default, the latest release will be installed, one can choose to install a specific release or even the master version.

```
$ spack install starpu@master
$ spack install starpu@1.3.5
```

We strongly advise reading the detailed reference manual at https://spack.readthedocs.io/en/latest/getting_started.html

5.2.2 Using a Module

On some clusters, StarPU is provided as a module, for example on the Jean Zay cluster. The information is available at http://www.idris.fr/jean-zay/cpu/jean-zay-cpu-starpu.html

5.3 Building from Source

StarPU can be built and installed by the standard means of the GNU autotools. The following chapter is intended to briefly remind how these tools can be used to install StarPU.

5.3.1 Optional Dependencies

The `hwloc` (http://www.open-mpi.org/software/hwloc) topology discovery library is not mandatory to use StarPU, but strongly recommended. It allows for topology aware scheduling, which improves performance. `hwloc` is available in major free operating system distributions, and for most operating systems. Make sure to not only install a `hwloc` or `libhwloc` package, but also `hwloc-devel` or `libhwloc-dev` to have `hwloc` headers etc.
If `libhwloc` is installed in a standard location, no option is required, it will be detected automatically, otherwise --with-hwloc=<directory> should be used to specify its location.
If `libhwloc` is not available on your system, the option --without-hwloc should be explicitly given when calling the script `configure`.

5.3.2 Getting Sources

StarPU's sources can be obtained from the download page of the StarPU website (https://starpu.gitlabpages.inria.fr/files/).
All releases and the development tree of StarPU are freely available on StarPU SCM server under the LGPL license. Some releases are available under the BSD license.
The latest release can be downloaded from the StarPU download page (https://starpu.gitlabpages.inria.fr/files/).
The latest nightly snapshot can be downloaded from the StarPU website (https://starpu.gitlabpages.inria.fr/files/testing/).
And finally, the current development version is also accessible via git. It should only be used if you need the very latest changes (i.e. less than a day old!).

```
$ git clone git@gitlab.inria.fr:starpu/starpu.git
```

5.3.3 Configuring StarPU

Running `autogen.sh` is not necessary when using the tarball releases of StarPU. However, when using the source code from the git repository, you first need to generate the script `configure` and the different Makefiles. This requires the availability of `autoconf` and `automake` >= 2.60.

```
$ ./autogen.sh
```

You then need to configure StarPU. Details about options that are useful to give to `configure` are given in Compilation Configuration.

```
$ ./configure
```

If `configure` does not detect some software or produces errors, please make sure to post the contents of the file `config.log` when reporting the issue.
By default, the files produced during the compilation are placed in the source directory. As the compilation generates a lot of files, it is advised to put them all in a separate directory. It is then easier to clean up, and this allows to compile several configurations out of the same source tree. To do so, simply enter the directory where you want the compilation to produce its files, and invoke the script `configure` located in the StarPU source directory.

```
$ mkdir build
$ cd build
$ ../configure
```

By default, StarPU will be installed in `/usr/local/bin`, `/usr/local/lib`, etc. You can specify an installation prefix other than `/usr/local` using the option `-prefix`, for instance:

```
$ ../configure --prefix=$HOME/starpu
```

5.3.4 Building StarPU

```
$ make
```

Once everything is built, you may want to test the result. An extensive set of regression tests is provided with StarPU. Running the tests is done by calling `make check`. These tests are run every night and the result from the main profile is publicly available (`https://starpu.gitlabpages/files/testing/master/`).

```
$ make check
```

5.3.5 Installing StarPU

In order to install StarPU at the location which was specified during configuration:

```
$ make install
```

If you have let StarPU install in `/usr/local/`, you additionally need to run

```
$ sudo ldconfig
```

so the libraries can be found by the system.
Libtool interface versioning information are included in libraries names (`libstarpu-1.4.so`, `libstarpumpi-1.4.so` and `libstarpufft-1.4.so`).

Chapter 6

Compilation Configuration

The behavior of the StarPU library and tools may be tuned thanks to the following configure options.

6.1 Common Configuration

–enable-debug Enable debugging messages.

–enable-spinlock-check Enable checking that spinlocks are taken and released properly.

–enable-fast Disable assertion checks, which saves computation time.

–enable-verbose Increase the verbosity of the debugging messages. This can be disabled at runtime by setting the environment variable STARPU_SILENT to any value. `–enable-verbose=extra` increase even more the verbosity.

```
$ STARPU_SILENT=1 ./vector_scal
```

–enable-coverage Enable flags for the coverage tool `gcov`.

–enable-quick-check Specify tests and examples should be run on a smaller data set, i.e allowing a faster execution time

–enable-long-check Enable some exhaustive checks which take a really long time.

–enable-new-check Enable new testcases which are known to fail.

–with-hwloc Specify `hwloc` should be used by StarPU. `hwloc` should be found by the means of the tool `pkg-config`.

–with-hwloc=`prefix` Specify `hwloc` should be used by StarPU. `hwloc` should be found in the directory specified by `prefix`

–without-hwloc Specify `hwloc` should not be used by StarPU.

–disable-build-doc Disable the creation of the documentation. This should be done on a machine which does not have the tools `doxygen` and `latex` (plus the packages `latex-xcolor` and `texlive-latex-extra`).

–enable-build-doc-pdf By default, only the HTML documentation is generated. Use this option to also enable the generation of the PDF documentation. This should be done on a machine which does have the tools `doxygen` and `latex` (plus the packages `latex-xcolor` and `texlive-latex-extra`).

–enable-icc Enable the compilation of specific ICC examples. StarPU itself will not be compiled with ICC unless specified with `CC=icc`

–disable-icc Disable the usage of the ICC compiler. Otherwise, when a ICC compiler is found, some specific ICC examples are compiled as explained above.

–with-check-flags Specify flags which will be given to C, CXX and Fortran compilers when valid

Additionally, the script `configure` recognize many variables, which can be listed by typing `./configure -help`. For example, `./configure NVCCFLAGS="-arch sm_20"` adds a flag for the compilation of CUDA kernels, and `NVCC_CC=gcc-5` allows to change the C++ compiler used by nvcc.

6.2 Configuring Workers

–enable-data-locality-enforce Enable data locality enforcement when picking up a worker to execute a task. This mechanism is by default disabled.

–enable-blocking-drivers By default, StarPU keeps CPU workers awake permanently, for better reactivity. This option makes StarPU put CPU workers to real sleep when there are not enough tasks to compute.

–enable-worker-callbacks If blocking drivers are enabled, enable callbacks to notify an external resource manager about workers going to sleep and waking up.

–enable-maxcpus=`count` Use at most `count` CPU cores. This information is then available as the macro STARPU_MAXCPUS.

The default value is `auto`. it allows StarPU to automatically detect the number of CPUs on the build machine. This should not be used if the running host has a larger number of CPUs than the build machine.

–enable-maxnumanodes=`count` Use at most `count` NUMA nodes. This information is then available as the macro STARPU_MAXNUMANODES.

The default value is `auto`. it allows StarPU to automatically detect the number of NUMA nodes on the build machine. This should not be used if the running host has a larger number of NUMA nodes than the build machine.

–disable-cpu Disable the use of CPUs of the machine. Only GPUs etc. will be used.

–enable-maxcudadev=`count` Use at most `count` CUDA devices. This information is then available as the macro STARPU_MAXCUDADEVS.

–disable-cuda Disable the use of CUDA, even if a valid CUDA installation was detected.

–with-cuda-dir=`prefix` Search for CUDA under `prefix`, which should notably contain the file `include/cuda.h`.

–with-cuda-include-dir=`dir` Search for CUDA headers under `dir`, which should notably contain the file `cuda.h`. This defaults to `/include` appended to the value given to --with-cuda-dir.

–with-cuda-lib-dir=`dir` Search for CUDA libraries under `dir`, which should notably contain the CUDA shared libraries—e.g., `libcuda.so`. This defaults to `/lib` appended to the value given to --with-cuda-dir.

–disable-cuda-memcpy-peer Explicitly disable peer transfers when using CUDA 4.0.

–enable-maxopencldev=`count` Use at most `count` OpenCL devices. This information is then available as the macro STARPU_MAXOPENCLDEVS.

–disable-opencl Disable the use of OpenCL, even if the SDK is detected.

–with-opencl-dir=`prefix` Search for an OpenCL implementation under `prefix`, which should notably contain `include/CL/cl.h` (or `include/OpenCL/cl.h` on Mac OS).

–with-opencl-include-dir=`dir` Search for OpenCL headers under `dir`, which should notably contain `CL/cl.h` (or `OpenCL/cl.h` on Mac OS). This defaults to `/include` appended to the value given to --with-opencl-dir.

–with-opencl-lib-dir=`dir` Search for an OpenCL library under `dir`, which should notably contain the OpenCL shared libraries—e.g. `libOpenCL.so`. This defaults to `/lib` appended to the value given to --with-opencl-dir.

–enable-opencl-simulator Enable considering the provided OpenCL implementation as a simulator, i.e. use the kernel duration returned by OpenCL profiling information as wallclock time instead of the actual measured real time. This requires the SimGrid support.

–enable-maximplementations=`count` Allow for at most `count` codelet implementations for the same target device. This information is then available as the macro STARPU_MAXIMPLEMENTATIONS macro.

–enable-max-sched-ctxs=`count` Allow for at most `count` scheduling contexts This information is then available as the macro STARPU_NMAX_SCHED_CTXS.

–disable-asynchronous-copy Disable asynchronous copies between CPU and GPU devices. The AMD implementation of OpenCL is known to fail when copying data asynchronously. When using this implementation, it is therefore necessary to disable asynchronous data transfers.

–disable-asynchronous-cuda-copy Disable asynchronous copies between CPU and CUDA devices.

–disable-asynchronous-opencl-copy Disable asynchronous copies between CPU and OpenCL devices. The AMD implementation of OpenCL is known to fail when copying data asynchronously. When using this implementation, it is therefore necessary to disable asynchronous data transfers.

–disable-asynchronous-hip-copy Disable asynchronous copies between CPU and HIP devices.

–disable-asynchronous-mpi-master-slave-copy Disable asynchronous copies between CPU and MPI Slave devices.

–disable-asynchronous-tcpip-master-slave-copy Disable asynchronous copies between CPU and MPI Slave devices.

–disable-asynchronous-fpga-copy Disable asynchronous copies between CPU and Maxeler FPGA devices.

–enable-maxnodes=`count` Use at most `count` memory nodes. This information is then available as the macro STARPU_MAXNODES. Reducing it allows to considerably reduce memory used by StarPU data structures.

–with-max-fpga=`dir` Enable the Maxeler FPGA driver support, and optionally specify the location of the Maxeler FPGA library.

–disable-asynchronous-max-fpga-copy Disable asynchronous copies between CPU and Maxeler FPGA devices.

6.3 Extension Configuration

–enable-starpupy Enable the StarPU Python Interface (Python Interface)

–enable-python-multi-interpreter Enable the use of multiple interpreters in the StarPU Python Interface (Multiple Interpreters)

–disable-mpi Disable the build of libstarpumpi. By default, it is enabled when MPI is found.

–enable-mpi Enable the build of libstarpumpi. This is necessary when using Simgrid+MPI.

–with-mpicc=`path` Use the compiler `mpicc` at `path`, for StarPU-MPI. (MPI Support).

–enable-mpi-pedantic-isend Before performing any MPI communication, StarPU-MPI waits for the data to be available in the main memory of the node submitting the request. For send communications, data is acquired with the mode STARPU_R. When enabling the pedantic mode, data are instead acquired with the STARPU_RW which thus ensures that there is not more than 1 concurrent `MPI_Isend` calls accessing the data and StarPU does not read from it from tasks during the communication.

–enable-mpi-master-slave Enable the MPI Master-Slave support. By default, it is disabled.

–enable-mpi-verbose Increase the verbosity of the MPI debugging messages. This can be disabled at runtime by setting the environment variable STARPU_SILENT to any value. `-enable-mpi-verbose=extra` increase even more the verbosity.

```
$ STARPU_SILENT=1 mpirun -np 2 ./insert_task
```

–enable-mpi-ft Enable the MPI checkpoint mechanism. See MPI Fault Tolerance Support

–enable-mpi-ft-stats Enable the statistics for the MPI checkpoint mechanism. See MPI Fault Tolerance Support

–enable-tcpip-master-slave Enable the TCP/IP Master-Slave support (TCP/IP Support). By default, it is disabled.

–enable-nmad Enable the NewMadeleine implementation for StarPU-MPI. See Using the NewMadeleine communication library for more details.

–disable-fortran Disable the fortran extension. By default, it is enabled when a fortran compiler is found.

–disable-socl Disable the SOCL extension (SOCL OpenCL Extensions). By default, it is enabled when an OpenCL implementation is found.

–enable-openmp Enable OpenMP Support (The StarPU OpenMP Runtime Support (SORS))

–enable-openmp-llvm Enable LLVM OpenMP Support (Example: An OpenMP LLVM Support)

–enable-bubble Enable Hierarchical dags support (Hierarchical DAGS)

–enable-parallel-worker Enable parallel worker support (Creating Parallel Workers On A Machine)

–enable-eclipse-plugin Enable the StarPU Eclipse Plugin. See StarPU Eclipse Plugin to know how to install Eclipse.

6.4 Advanced Configuration

–enable-perf-debug Enable performance debugging through gprof.

–enable-model-debug Enable performance model debugging.

–enable-fxt-lock Enable additional trace events which describes locks behaviour. This is however extremely heavy and should only be enabled when debugging insides of StarPU.

–enable-maxbuffers Define the maximum number of buffers that tasks will be able to take as parameters, then available as the macro STARPU_NMAXBUFS.

–enable-fxt-max-files=`count` Use at most `count` mpi nodes fxt files for generating traces. This information is then available as the macro STARPU_FXT_MAX_FILES. This information is used by FxT tools when considering multi node traces. Default value is 64.

–enable-allocation-cache Enable the use of a data allocation cache to avoid the cost of it with CUDA. Still experimental.

–enable-opengl-render Enable the use of OpenGL for the rendering of some examples.

–enable-blas-lib=`prefix` Specify the blas library to be used by some of the examples. Libraries available :

- `none` [default] : no BLAS library is used
- `atlas`: use ATLAS library
- `goto`: use GotoBLAS library
- `openblas`: use OpenBLAS library
- `mkl`: use MKL library (you may need to set specific `CFLAGS` and `LDFLAGS` with –with-mkl-cflags and –with-mkl-ldflags)

–enable-leveldb Enable linking with LevelDB if available

–enable-hdf5 Enable building HDF5 support.

–with-hdf5-include-dir=`path` Specify the directory where is stored the header file `hdf5.h`.

–with-hdf5-lib-dir=`path` Specify the directory where is stored the library `hdf5`.

–disable-starpufft Disable the build of libstarpufft, even if `fftw` or `cuFFT` is available.

–enable-starpufft-examples Enable the compilation and the execution of the libstarpufft examples. By default, they are neither compiled nor checked.

–with-fxt=`prefix` Search for FxT under `prefix`. FxT (`http://savannah.nongnu.org/projects/fkt`) is used to generate traces of scheduling events, which can then be rendered them using ViTE (Off-linePerformanceFeedback). `prefix` should notably contain `include/fxt/fxt.h`.

–with-perf-model-dir=`dir` Store performance models under `dir`, instead of the current user's home.

–with-goto-dir=`prefix` Search for GotoBLAS under `prefix`, which should notably contain `libgoto.so` or `libgoto2.so`.

–with-atlas-dir=`prefix` Search for ATLAS under `prefix`, which should notably contain `include/cblas.h`.

–with-mkl-cflags=`cflags` Use `cflags` to compile code that uses the MKL library.

–with-mkl-ldflags=`ldflags` Use `ldflags` when linking code that uses the MKL library. Note that the MKL website (`http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor/`) provides a script to determine the linking flags.

–disable-glpk Disable the use of `libglpk` for computing area bounds.

–disable-build-tests Disable the build of tests.

–disable-build-examples Disable the build of examples.

–enable-sc-hypervisor Enable the Scheduling Context Hypervisor plugin (Scheduling Context Hypervisor). By default, it is disabled.

–enable-memory-stats Enable memory statistics (Memory Feedback).

–enable-simgrid Enable simulation of execution in SimGrid, to allow easy experimentation with various numbers of cores and GPUs, or amount of memory, etc. Experimental.

The path to SimGrid can be specified through the `SIMGRID_CFLAGS` and `SIMGRID_LIBS` environment variables, for instance:

```
export SIMGRID_CFLAGS="-I/usr/local/simgrid/include"
export SIMGRID_LIBS="-L/usr/local/simgrid/lib -lsimgrid"
```

–with-simgrid-dir Similar to the option --enable-simgrid but also allows to specify the location to the SimGrid library.

–with-simgrid-include-dir Similar to the option --enable-simgrid but also allows to specify the location to the SimGrid include directory.

–with-simgrid-lib-dir Similar to the option --enable-simgrid but also allows to specify the location to the SimGrid lib directory.

–with-smpirun=`path` Use the smpirun at `path`

–enable-simgrid-mc Enable the Model Checker in simulation of execution in SimGrid, to allow exploring various execution paths.

–enable-calibration-heuristic Allow to set the maximum authorized percentage of deviation for the history-based calibrator of StarPU. A correct value of this parameter must be in [0..100]. The default value of this parameter is 10. Experimental.

–enable-mlr Allow to enable multiple linear regression models (see Performance Model Example)

–enable-mlr-system-blas Allow to make multiple linear regression models use the system-provided BLAS for dgels (see Performance Model Example)

Chapter 7

Execution Configuration Through Environment Variables

The StarPU library and tools's behavior can be tuned using the following environment variables. To access these variables, you can use the provided functions.

- starpu_getenv() retrieves the value of an environment variable.
- starpu_get_env_string_var_default() retrieves the value of an environment variable as a string. If the variable is not set, you can provide a default value.
- starpu_get_env_size_default() retrieves the value of an environment variable as a size in bytes, or a default value if the environment variable is not set.

These functions allow to fine-tune the behavior of StarPU according to your preferences and requirements by leveraging environment variables.

7.1 Configuring Workers

7.1.1 General Configuration

STARPU_WORKERS_NOBIND Setting it to non-zero will prevent StarPU from binding its threads to CPUs. This is for instance useful when running the test suite in parallel.

STARPU_WORKERS_GETBIND By default StarPU uses the OS-provided CPU binding to determine how many and which CPU cores it should use. This is notably useful when running several StarPU-MPI processes on the same host, to let the MPI launcher set the CPUs to be used. Default value is 1.

If that binding is erroneous (e.g. because the job scheduler binds to just one core of the allocated cores), you can set STARPU_WORKERS_GETBIND to 0 to make StarPU use all cores of the machine.

STARPU_WORKERS_CPUID Passing an array of integers in STARPU_WORKERS_CPUID specifies on which logical CPU the different workers should be bound. For instance, if `STARPU_WORKERS_CPUID="0 1 4 5"`, the first worker will be bound to logical CPU #0, the second CPU worker will be bound to logical CPU #1 and so on. Note that the logical ordering of the CPUs is either determined by the OS, or provided by the library `hwloc` in case it is available. Ranges can be provided: for instance, `STARPU_WORKERS_CPUID="1-3 5"` will bind the first three workers on logical CPUs #1, #2, and #3, and the fourth worker on logical CPU #5. Unbound ranges can also be provided: `STARPU_WORKERS_CPUID="1-"` will bind the workers starting from logical CPU #1 up to last CPU.

Note that the first workers correspond to the CUDA workers, then come the OpenCL workers, and finally the CPU workers. For example, if we have `STARPU_NCUDA=1`, `STARPU_NOPENCL=1`, `STARPU_NCPU=2` and `STARPU_WORKERS_CPUID="0 2 1 3"`, the CUDA device will be controlled by logical CPU #0, the OpenCL device will be controlled by logical CPU #2, and the logical CPUs #1 and #3 will be used by the CPU workers.

If the number of workers is larger than the array given in STARPU_WORKERS_CPUID, the workers are bound to the logical CPUs in a round-robin fashion: if `STARPU_WORKERS_CPUID="0 1"`, the first and the third (resp. second and fourth) workers will be put on CPU #0 (resp. CPU #1).

This variable is ignored if the field starpu_conf::use_explicit_workers_bindid passed to starpu_init() is set.

Setting STARPU_WORKERS_CPUID or STARPU_WORKERS_COREID overrides the binding provided by the job scheduler, as described for STARPU_WORKERS_GETBIND.

STARPU_WORKERS_COREID Same as STARPU_WORKERS_CPUID, but bind the workers to cores instead of PUs (hyperthreads).

STARPU_NTHREADS_PER_CORE Specify how many threads StarPU should run on each core. The default is 1 because kernels are usually already optimized for using a full core. Setting this to e.g. 2 instead allows exploiting hyperthreading.

STARPU_MAIN_THREAD_BIND Tell StarPU to bind the thread that calls starpu_initialize() to a reserved CPU, subtracted from the CPU workers.

STARPU_MAIN_THREAD_CPUID Tell StarPU to bind the thread that calls starpu_initialize() to the given CPU ID (using logical numbering).

STARPU_MAIN_THREAD_COREID Same as STARPU_MAIN_THREAD_CPUID, but bind the thread that calls starpu_initialize() to the given core (using logical numbering), instead of the PU (hyperthread).

STARPU_WORKER_TREE Define to 1 to enable the tree iterator in schedulers.

STARPU_SINGLE_COMBINED_WORKER Tell StarPU to create several workers which won't be able to work concurrently. It will by default create combined workers, which size goes from 1 to the total number of CPU workers in the system. STARPU_MIN_WORKERSIZE and STARPU_MAX_WORKERSIZE can be used to change this default.

STARPU_MIN_WORKERSIZE Specify the minimum size of the combined workers. Default value is 2.

STARPU_MAX_WORKERSIZE Specify the minimum size of the combined workers. Default value is the number of CPU workers in the system.

STARPU_SYNTHESIZE_ARITY_COMBINED_WORKER Specify how many elements are allowed between combined workers created from `hwloc` information. For instance, in the case of sockets with 6 cores without shared L2 caches, if STARPU_SYNTHESIZE_ARITY_COMBINED_WORKER is set to 6, no combined worker will be synthesized beyond one for the socket and one per core. If it is set to 3, 3 intermediate combined workers will be synthesized, to divide the socket cores into 3 chunks of 2 cores. If it set to 2, 2 intermediate combined workers will be synthesized, to divide the socket cores into 2 chunks of 3 cores, and then 3 additional combined workers will be synthesized, to divide the former synthesized workers into a bunch of 2 cores, and the remaining core (for which no combined worker is synthesized since there is already a normal worker for it).

Default value is 2, thus makes StarPU tend to build binary trees of combined workers.

STARPU_DISABLE_ASYNCHRONOUS_COPY Disable asynchronous copies between CPU and GPU devices. The AMD implementation of OpenCL is known to fail when copying data asynchronously. When using this implementation, it is therefore necessary to disable asynchronous data transfers. One can call starpu_asynchronous_copy_disabled() to check whether asynchronous data transfers between CPU and accelerators are disabled.

See also STARPU_DISABLE_ASYNCHRONOUS_CUDA_COPY and STARPU_DISABLE_ASYNCHRONOUS_OPENCL_COP

STARPU_EXPECTED_TRANSFER_TIME_WRITEBACK Set to 1 to make task transfer time estimations artificially include the time that will be needed to write back data to the main memory.

STARPU_DISABLE_PINNING Disable (1) or Enable (0) pinning host memory allocated through starpu_malloc(), starpu_memory_pin() and friends. Default value is Enable. This permits to test the performance effect of memory pinning.

STARPU_BACKOFF_MIN Set minimum exponential backoff of number of cycles to pause when spinning. Default value is 1.

STARPU_BACKOFF_MAX Set maximum exponential backoff of number of cycles to pause when spinning. Default value is 32.

STARPU_SINK Defined internally by StarPU when running in master slave mode.

STARPU_ENABLE_MAP Disable (0) or Enable (1) support for memory mapping between memory nodes. The default is Disabled. One can call starpu_map_enabled() to check whether memory mapping support between memory nodes is enabled.

STARPU_DATA_LOCALITY_ENFORCE Enable (1) or Disable(0) data locality enforcement when picking up a worker to execute a task. Default value is Disable.

7.1.2 CPU Workers

STARPU_NCPU Specify the number of CPU workers (thus not including workers dedicated to control accelerators). Note that by default, StarPU will not allocate more CPU workers than there are physical CPUs, and that some CPUs are used to control the accelerators.

STARPU_RESERVE_NCPU Specify the number of CPU cores that should not be used by StarPU, so the application can use starpu_get_next_bindid() and starpu_bind_thread_on() to bind its own threads.

This option is ignored if STARPU_NCPU or starpu_conf::ncpus is set.

STARPU_NCPUS Deprecated. You should use STARPU_NCPU.

7.1.3 CUDA Workers

STARPU_NCUDA Specify the number of CUDA devices that StarPU can use. If STARPU_NCUDA is lower than the number of physical devices, it is possible to select which GPU devices should be used by the means of the environment variable STARPU_WORKERS_CUDAID. By default, StarPU will create as many CUDA workers as there are GPU devices.

STARPU_NWORKER_PER_CUDA Specify the number of workers per CUDA device, and thus the number of kernels which will be concurrently running on the devices, i.e. the number of CUDA streams. Default value is 1.

STARPU_CUDA_THREAD_PER_WORKER Specify whether the cuda driver should use one thread per stream (1) or to use a single thread to drive all the streams of the device or all devices (0), and STARPU_CUDA_THREAD_PER_DEV determines whether is it one thread per device or one thread for all devices. Default value is 0. Setting it to 1 is contradictory with setting STARPU_CUDA_THREAD_PER_DEV.

STARPU_CUDA_THREAD_PER_DEV Specify whether the cuda driver should use one thread per device (1) or to use a single thread to drive all the devices (0). Default value is 1. It does not make sense to set this variable if STARPU_CUDA_THREAD_PER_WORKER is set to to 1 (since STARPU_CUDA_THREAD_PER_DEV is then meaningless).

STARPU_CUDA_PIPELINE Specify how many asynchronous tasks are submitted in advance on CUDA devices. This for instance permits to overlap task management with the execution of previous tasks, but it also allows concurrent execution on Fermi cards, which otherwise bring spurious synchronizations. Default value is 2. Setting the value to 0 forces a synchronous execution of all tasks.

STARPU_WORKERS_CUDAID Select which CUDA devices should be used to run CUDA workers (similarly to the STARPU_WORKERS_CPUID environment variable). On a machine equipped with 4 GPUs, setting `STARPU_WORKERS_CUDAID="1 3"` and `STARPU_NCUDA=2` specifies that 2 CUDA workers should be created, and that they should use CUDA devices #1 and #3 (the logical ordering of the devices is the one reported by CUDA).

This variable is ignored if the field starpu_conf::use_explicit_workers_cuda_gpuid passed to starpu_init() is set.

STARPU_DISABLE_ASYNCHRONOUS_CUDA_COPY Disable asynchronous copies between CPU and CUDA devices. One can call starpu_asynchronous_cuda_copy_disabled() to check whether asynchronous data transfers between CPU and CUDA accelerators are disabled.

See also STARPU_DISABLE_ASYNCHRONOUS_COPY and STARPU_DISABLE_ASYNCHRONOUS_OPENCL_COPY.

STARPU_ENABLE_CUDA_GPU_GPU_DIRECT Enable (1) or Disable (0) direct CUDA transfers from GPU to GPU, without copying through RAM. Default value is Enable. This permits to test the performance effect of GPU-Direct.

STARPU_CUDA_ONLY_FAST_ALLOC_OTHER_MEMNODES Specify if CUDA workers should do only fast allocations when running the datawizard progress of other memory nodes. This will pass the internal value _STARPU_DATAWIZARD_ONLY_FAST_ALLOC to allocation methods. Default value is 0, allowing CUDA workers to do slow allocations.

This can also be specified with starpu_conf::cuda_only_fast_alloc_other_memnodes.

7.1.4 OpenCL Workers

STARPU_NOPENCL Specify the number of OpenCL devices that StarPU can use. If STARPU_NOPENCL is lower than the number of physical devices, it is possible to select which GPU devices should be used by the means of the environment variable STARPU_WORKERS_OPENCLID. By default, StarPU will create as many OpenCL workers as there are GPU devices.

Note that by default StarPU will launch CUDA workers on GPU devices. You need to disable CUDA to allow the creation of OpenCL workers.

STARPU_WORKERS_OPENCLID Select which GPU devices should be used to run OpenCL workers (similarly to the STARPU_WORKERS_CPUID environment variable) On a machine equipped with 4 GPUs, setting `STARPU_WORKERS_OPENCLID="1 3"` and `STARPU_NOPENCL=2` specifies that 2 OpenCL workers should be created, and that they should use GPU devices #1 and #3.

This variable is ignored if the field starpu_conf::use_explicit_workers_opencl_gpuid passed to starpu_init() is set.

STARPU_OPENCL_PIPELINE Specify how many asynchronous tasks are submitted in advance on OpenCL devices. This for instance permits to overlap task management with the execution of previous tasks, but it also allows concurrent execution on Fermi cards, which otherwise bring spurious synchronizations. Default value is 2. Setting the value to 0 forces a synchronous execution of all tasks.

STARPU_OPENCL_ON_CPUS Specify that OpenCL workers can also be run on CPU devices. By default, the OpenCL driver only enables GPU devices.

STARPU_OPENCL_ONLY_ON_CPUS Specify that OpenCL workers can ONLY be run on CPU devices. By default, the OpenCL driver enables GPU devices.

STARPU_DISABLE_ASYNCHRONOUS_OPENCL_COPY Disable asynchronous copies between CPU and OpenCL devices. The AMD implementation of OpenCL is known to fail when copying data asynchronously. When using this implementation, it is therefore necessary to disable asynchronous data transfers. One can call starpu_asynchronous_opencl_copy_disabled() to check whether asynchronous data transfers between CPU and OpenCL accelerators are disabled.

See also STARPU_DISABLE_ASYNCHRONOUS_COPY and STARPU_DISABLE_ASYNCHRONOUS_CUDA_COPY.

7.1.5 Maxeler FPGA Workers

STARPU_NMAX_FPGA Specify the number of Maxeler FPGA devices that StarPU can use. If STARPU_NMAX_FPGA is lower than the number of physical devices, it is possible to select which Maxeler FPGA devices should be used by the means of the environment variable STARPU_WORKERS_MAX_FPGAID. By default, StarPU will create as many Maxeler FPGA workers as there are GPU devices.

STARPU_WORKERS_MAX_FPGAID Select which Maxeler FPGA devices should be used to run Maxeler FPGA workers (similarly to the STARPU_WORKERS_CPUID environment variable). On a machine equipped with 4 Maxeler FPGAs, setting `STARPU_WORKERS_MAX_FPGAID="1 3"` and `STARPU_NMAX_FPGA=2` specifies that 2 Maxeler FPGA workers should be created, and that they should use Maxeler FPGA devices #1 and #3 (the logical ordering of the devices is the one reported by the Maxeler stack).

STARPU_DISABLE_ASYNCHRONOUS_MAX_FPGA_COPY Disable asynchronous copies between CPU and Maxeler FPGA devices. One can call starpu_asynchronous_max_fpga_copy_disabled() to check whether asynchronous data transfers between CPU and Maxeler FPGA devices are disabled.

7.1.6 MPI Master Slave Workers

STARPU_NMPI_MS Specify the number of MPI master slave devices that StarPU can use.

STARPU_NMPIMSTHREADS Specift the number of threads to use on the MPI Slave devices.

STARPU_MPI_MS_MULTIPLE_THREAD Specify whether the master should use one thread per slave, or one thread for driver all slaves. Default value is 0.

STARPU_MPI_MASTER_NODE Specify the rank of the MPI process which will be the master. Default value is 0.

STARPU_DISABLE_ASYNCHRONOUS_MPI_MS_COPY Disable asynchronous copies between CPU and MPI Slave devices. One can call starpu_asynchronous_mpi_ms_copy_disabled() to check whether asynchronous data transfers between CPU and MPI Slave devices are disabled.

7.1.7 TCP/IP Master Slave Workers

STARPU_NTCPIP_MS Specify the number of TCP/IP master slave devices that StarPU can use.

STARPU_TCPIP_MS_SLAVES Specify the number of TCP/IP master slave processes that are expected to be run. This should be provided both to the master and to the slaves.

STARPU_TCPIP_MS_MASTER Specify (for slaves) the IP address of the master so they can connect to it. They will then automatically connect to each other.

STARPU_TCPIP_MS_PORT Specify the port of the master, for connexions between slaves and the master. Default value is 1234.

STARPU_NTCPIPMSTHREADS Specify the number of threads to use on the TCP/IP Slave devices.

STARPU_TCPIP_MS_MULTIPLE_THREAD Specify whether the master should use one thread per slave, or one thread for driver all slaves. Default value is 0.

STARPU_DISABLE_ASYNCHRONOUS_TCPIP_MS_COPY Disable asynchronous copies between CPU and TCP/IP Slave devices. One can call starpu_asynchronous_tcpip_ms_copy_disabled() to check whether asynchronous data transfers between CPU and TCP/IP Slave devices are disabled.

7.1.8 HIP Workers

STARPU_NHIP Specify the number of HIP devices that StarPU can use. If STARPU_NHIP is lower than the number of physical devices, it is possible to select which HIP devices should be used by the means of the environment variable STARPU_WORKERS_HIPID. By default, StarPU will create as many HIP workers as there are HIP devices.

STARPU_WORKERS_HPIID Select which HIP devices should be used to run HIP workers (similarly to the STARPU_WORKERS_HIPID environment variable). On a machine equipped with 4 HIP devices, setting `STARPU_WORKERS_HIPID="1 3"` and `STARPU_NHIP=2` specifies that 2 HIP workers should be created, and that they should use HIP devices #1 and #3.

This variable is ignored if the field starpu_conf::use_explicit_workers_hip_gpuid passed to starpu_init() is set.

STARPU_DISABLE_ASYNCHRONOUS_HIP_COPY Disable asynchronous copies between CPU and HIP devices. One can call starpu_asynchronous_hip_copy_disabled() to check whether asynchronous data transfers between CPU and HIP accelerators are disabled.

7.1.9 MPI Configuration

STARPU_MPI_THREAD_CPUID Tell StarPU to bind its MPI thread to the given CPU id, subtracted from the CPU workers (unless STARPU_NCPU is defined).

Default value is -1, it will let StarPU allocate a CPU.

STARPU_MPI_THREAD_COREID Same as STARPU_MPI_THREAD_CPUID, but bind the MPI thread to the given core ID, instead of the PU (hyperthread).

STARPU_MPI_NOBIND Setting it to non-zero will prevent StarPU from binding the MPI to a separate core. This is for instance useful when running the testsuite on a single system.

STARPU_MPI_GPUDIRECT Enable (1) or disable (0) MPI GPUDirect support. Default value (-1) is to enable if available. If STARPU_MPI_GPUDIRECT is explicitly set to 1, StarPU-MPI will warn if MPI does not provide the GPUDirect support.

STARPU_MPI_PSM2 This variable allows to supercede PSM2 detection when asking for MPI GPUDirect support. This is helpful when using old intel compilers, for which PSM2 detection is always true. The default (1) is to enable it. If PSM2 is detected whereas it should not be, this variable can be set to 0.

STARPU_MPI_REDUX_ARITY_THRESHOLD The arity of the automatically-detected reduction trees follows the following rule: when the data to be reduced is of small size a flat tree is unrolled i.e. all the contributing nodes send their contribution to the root of the reduction. When the data to be reduced is of big size, a binary tree is used instead. The default threshold between flat and binary tree is 1024 bytes. By setting the environment variable with a negative value, all the automatically detected reduction trees will use flat trees. If this value is set to 0, then binary trees will always be selected. Otherwise, the setup value replaces the default 1024.

7.2 Configuring The Scheduling Engine

STARPU_SCHED Select the scheduling policy from those proposed by StarPU: work random, stealing, greedy, with performance models, etc.

Use `STARPU_SCHED=help` to get the list of available schedulers.

STARPU_SCHED_LIB Specify the location of a dynamic library to choose a user-defined scheduling policy. See Using a New Scheduling Policy for more information.

STARPU_MIN_PRIO Set the minimum priority used by priorities-aware schedulers. The flag can also be set through the field starpu_conf::global_sched_ctx_min_priority.

STARPU_MAX_PRIO Set the maximum priority used by priorities-aware schedulers. The flag can also be set through the field starpu_conf::global_sched_ctx_max_priority.

STARPU_CALIBRATE Set to 1 to calibrate the performance models during the execution. Set to 2 to drop the previous values and restart the calibration from scratch. Set to 0 to disable calibration, this is the default behaviour.

Note: this currently only applies to `dm` and `dmda` scheduling policies.

STARPU_CALIBRATE_MINIMUM Define the minimum number of calibration measurements that will be made before considering that the performance model is calibrated. Default value is 10.

STARPU_BUS_CALIBRATE Set to 1 to recalibrate the bus during initialization.

STARPU_PREFETCH Enable (1) or disable (0) data prefetching. Default value is Enable.

If prefetching is enabled, when a task is scheduled to be executed e.g. on a GPU, StarPU will request an asynchronous transfer in advance, so that data is already present on the GPU when the task starts. As a result, computation and data transfers are overlapped.

STARPU_SCHED_ALPHA To estimate the cost of a task StarPU takes into account the estimated computation time (obtained thanks to performance models). The alpha factor is the coefficient to be applied to it before adding it to the communication part.

STARPU_SCHED_BETA To estimate the cost of a task StarPU takes into account the estimated data transfer time (obtained thanks to performance models). The beta factor is the coefficient to be applied to it before adding it to the computation part.

STARPU_SCHED_GAMMA Define the execution time penalty of a joule (Energy-based Scheduling).

STARPU_SCHED_READY For a modular scheduler with sorted queues below the decision component, workers pick up a task which has most of its data already available. Setting this to 0 disables this.

STARPU_SCHED_SORTED_ABOVE For a modular scheduler with queues above the decision component, it is usually sorted by priority. Setting this to 0 disables this.

STARPU_SCHED_SORTED_BELOW For a modular scheduler with queues below the decision component, they are usually sorted by priority. Setting this to 0 disables this.

STARPU_IDLE_POWER Define the idle power of the machine (Energy-based Scheduling).

STARPU_PROFILING Enable on-line performance monitoring (Enabling On-line Performance Monitoring).

STARPU_CODELET_PROFILING Enable on-line performance monitoring of codelets (Per-codelet Feedback). (enabled by default)

STARPU_ENERGY_PROFILING Enable on-line energy monitoring of tasks (Per-codelet Feedback). (disabled by default)

STARPU_PROF_PAPI_EVENTS Specify which PAPI events should be recorded in the trace (PAPI counters).

7.3 Configuring The Heteroprio Scheduler

7.3.1 Configuring LAHeteroprio

STARPU_HETEROPRIO_USE_LA Enable the locality aware mode of Heteroprio which guides the distribution of tasks to workers in order to reduce the data transfers between memory nodes.

STARPU_LAHETEROPRIO_PUSH Choose between the different push strategies for locality aware Heteroprio: `WORKER`, `LcS`, `LS_SDH`, `LS_SDH2`, `LS_SDHB`, `LC_SMWB`, `AUTO` (by default: AUTO). These are detailed in Using locality aware Heteroprio

STARPU_LAHETEROPRIO_S_[ARCH] [ARCH] Specify the number of memory nodes contained in an affinity group. An affinity group will be composed of the closest memory nodes to a worker of a given architecture, and this worker will look for tasks available inside these memory nodes, before considering stealing tasks outside this group. ARCH can be `CPU`, `CUDA`, `OPENCL`, `SCC`, `MPI_MS`, etc.

STARPU_LAHETEROPRIO_PRIO_STEP_[ARCH] [ARCH] Specify the number of buckets in the local memory node in which a worker will look for available tasks, before this worker starts looking for tasks in other memory nodes' buckets. ARCH indicates that this number is specific to a given arch which can be: `CPU`, `CUDA`, `OPENCL`, `SCC`, `MPI_MS`, etc.

7.3.2 Configuring AutoHeteroprio

STARPU_HETEROPRIO_USE_AUTO_CALIBRATION Enable the auto calibration mode of Heteroprio which assign priorities to tasks automatically

STARPU_HETEROPRIO_DATA_DIR Specify the path of the directory where Heteroprio stores data about program executions. By default, these are stored in the same directory used by perfmodel.

STARPU_HETEROPRIO_DATA_FILE Specify the filename where Heteroprio will save data about the current program's execution.

STARPU_HETEROPRIO_CODELET_GROUPING_STRATEGY Choose how Heteroprio groups similar tasks. It can be `0` to group the tasks with the same perfmodel or the same codelet's name if no perfmodel was assigned. Or, it could be `1` to group the tasks only by codelet's name.

STARPU_AUTOHETEROPRIO_PRINT_DATA_ON_UPDATE Enable the printing of priorities' data every time they get updated.

STARPU_AUTOHETEROPRIO_PRINT_AFTER_ORDERING Enable the printing of priorities' order for each architecture every time there's a reordering.

STARPU_AUTOHETEROPRIO_PRIORITY_ORDERING_POLICY Specify the heuristic which will be used to assign priorities automatically. It should be an integer between 0 and 27.

STARPU_AUTOHETEROPRIO_ORDERING_INTERVAL Specify the period (in number of tasks pushed), between priorities reordering operations.

STARPU_AUTOHETEROPRIO_FREEZE_GATHERING Disable data gathering from task executions.

7.4 Extensions

SOCL_OCL_LIB_OPENCL Set the location of the file `libOpenCL.so` of the OCL ICD implementation. The SOCL test suite is only run when SOCL_OCL_LIB_OPENCL is defined.

OCL_ICD_VENDORS Set the directory where ICD files are installed. This is useful when using SOCL with OpenCL ICD (`https://forge.imag.fr/projects/ocl-icd/`). Default directory is `/etc/OpenCL/vendors`. StarPU installs ICD files in the directory `$prefix/share/starpu/opencl/vendors`.

STARPU_COMM_STATS Deprecated. You should use STARPU_MPI_STATS.

STARPU_MPI_STATS Enable (!= 0) or Disable (0) communication statistics for starpumpi (Debugging MPI). Default value is Disable.

STARPU_MPI_CACHE Disable (0) or Enable (!= 0) communication cache for starpumpi (MPI Support). Default value is Enable.

STARPU_MPI_COMM Enable (1) communication trace for starpumpi (MPI Support). Also needs for StarPU to have been configured with the option --enable-verbose.

STARPU_MPI_CACHE_STATS Enable (1) statistics for the communication cache (MPI Support). Messages are printed on the standard output when data are added or removed from the received communication cache.

STARPU_MPI_PRIORITIES Disable (0) the use of priorities to order MPI communications (MPI Support).

STARPU_MPI_NDETACHED_SEND Set the number of send requests that StarPU-MPI will emit concurrently. Default value is 10. Setting it to 0 removes the limit of concurrent send requests.

STARPU_MPI_NREADY_PROCESS Set the number of requests that StarPU-MPI will submit to MPI before polling for termination of existing requests. Default value is 10. Setting it to 0 removes the limit: all requests to submit to MPI will be submitted before polling for termination of existing ones.

STARPU_MPI_FAKE_SIZE Setting to a number makes StarPU believe that there are as many MPI nodes, even if it was run on only one MPI node. This allows e.g. to simulate the execution of one of the nodes of a big cluster without actually running the rest. Of course, it does not provide computation results and timing.

STARPU_MPI_FAKE_RANK Setting to a number makes StarPU believe that it runs the given MPI node, even if it was run on only one MPI node. This allows e.g. to simulate the execution of one of the nodes of a big cluster without actually running the rest. Of course, it does not provide computation results and timing.

STARPU_MPI_COOP_SENDS Disable (0) dynamic collective operations: grouping same requests to different nodes until the data becomes available and then use a broadcast tree to execute requests.
By now, it is only supported with the NewMadeleine library (see Using the NewMadeleine communication library).

STARPU_MPI_RECV_WAIT_FINALIZE Disable (1) releasing the write acquire of receiving handles when data is received but the communication library still needs the data. Set to 0 by default to unlock as soon as possible tasks which only require a read access on the handle; write access will become possible for tasks when the communication library will not need the data anymore.
By now, it is only supported with the NewMadeleine library (see Using the NewMadeleine communication library).

STARPU_MPI_TRACE_SYNC_CLOCKS When `mpi_sync_clocks` is available, this library will be used to have more precise clock synchronization in traces coming from different nodes. However, the clock synchronization process can take some time (several seconds) and can be disabled by setting this variable to `0`. In that case, a less precise but faster synchronization will be used. See Tracing MPI applications for more details.

STARPU_MPI_DRIVER_CALL_FREQUENCY When set to a positive value, activates the interleaving of the execution of tasks with the progression of MPI communications (MPI Support). The starpu_mpi_init_conf() function must have been called by the application for that environment variable to be used. When set to 0, the MPI progression thread does not use at all the driver given by users, and only focuses on making MPI communications progress.

STARPU_MPI_DRIVER_TASK_FREQUENCY When set to a positive value, the interleaving of the execution of tasks with the progression of MPI communications mechanism to execute several tasks before checking communication requests again (MPI Support). The starpu_mpi_init_conf() function must have been called by the application for that environment variable to be used, and the STARPU_MPI_DRIVER_CALL_FREQUENCY environment variable set to a positive value.

STARPU_MPI_MEM_THROTTLE When set to a positive value, this makes the starpu_mpi_∗recv∗ functions block when the memory allocation required for network reception overflows the available main memory (as typically set by STARPU_LIMIT_CPU_MEM)

STARPU_MPI_EARLYDATA_ALLOCATE When set to 1, the MPI Driver will immediately allocate the data for early requests instead of issuing a data request and blocking. Default value is 0, issuing a data request. Because it is an early request and we do not know its real priority, the data request will assume STARPU_DEFAULT_PRIO. In cases where there are many data requests with priorities greater than STARPU_DEFAULT_PRIO the MPI drive could be blocked for long periods.

STARPU_SIMGRID When set to 1 (default value is 0), this makes StarPU check that it was really build with simulation support. This is convenient in scripts to avoid using a native version, that would try to update performance models...

STARPU_SIMGRID_TRANSFER_COST When set to 1 (which is the default value), data transfers (over PCI bus, typically) are taken into account in SimGrid mode.

STARPU_SIMGRID_CUDA_MALLOC_COST When set to 1 (which is the default value), CUDA malloc costs are taken into account in SimGrid mode.

STARPU_SIMGRID_CUDA_QUEUE_COST When set to 1 (which is the default value), CUDA task and transfer queueing costs are taken into account in SimGrid mode.

STARPU_PCI_FLAT When unset or set to 0, the platform file created for SimGrid will contain PCI bandwidths and routes.

STARPU_SIMGRID_CUDA_QUEUE_COST When unset or set to 1, simulate within SimGrid the GPU transfer queueing.

STARPU_MALLOC_SIMULATION_FOLD Define the size of the file used for folding virtual allocation, in MiB. Default value is 1, thus allowing 64GiB virtual memory when Linux's `sysctl vm.max_map_count` value is the default 65535.

STARPU_SIMGRID_TASK_SUBMIT_COST When set to 1 (which is the default value), task submission costs are taken into account in SimGrid mode. This provides more accurate SimGrid predictions, especially for the beginning of the execution.

STARPU_SIMGRID_TASK_PUSH_COST When set to 1 (which is the default value), task push costs are taken into account in SimGrid mode. This provides more accurate SimGrid predictions, especially with large dependency arities.

STARPU_SIMGRID_FETCHING_INPUT_COST When set to 1 (which is the default value), fetching input costs are taken into account in SimGrid mode. This provides more accurate SimGrid predictions, especially regarding data transfers.

STARPU_SIMGRID_SCHED_COST When set to 1 (0 is the default value), scheduling costs are taken into account in SimGrid mode. This provides more accurate SimGrid predictions, and allows studying scheduling overhead of the runtime system. However, it also makes simulation non-deterministic.

STARPUPY_MULTI_INTERPRETER Enable (1) or disable (0) multi interpreters in the StarPU Python interface (Multiple Interpreters). Default value is Disable.

STARPUPY_OWN_GIL Enable (1) or disable (0) using per-interpreter GIL (Python Parallelism). Default value is Disable for now, until python is fully ready for this.

7.5 Miscellaneous And Debug

STARPU_HOME Specify the main directory in which StarPU stores its configuration files. Default value is `$HOME` on Unix environments, and `$USERPROFILE` on Windows environments.

STARPU_PATH Only used on Windows environments. Specify the main directory in which StarPU is installed (Running a Basic StarPU Application on Microsoft Visual C)

STARPU_PERF_MODEL_DIR Specify the main directory in which StarPU stores its performance model files. Default value is `$STARPU_HOME/.starpu/sampling`. See Storing Performance Model Files for more details.

STARPU_PERF_MODEL_PATH Specify a list of directories separated with ':' in which StarPU stores its performance model files. See Storing Performance Model Files for more details.

STARPU_PERF_MODEL_HOMOGENEOUS_CPU When set to 0, StarPU will assume that CPU devices do not have the same performance, and thus use different performance models for them, thus making kernel calibration much longer, since measurements have to be made for each CPU core.

STARPU_PERF_MODEL_HOMOGENEOUS_CUDA When set to 1, StarPU will assume that all CUDA devices have the same performance, and thus share performance models for them, thus allowing kernel calibration to be much faster, since measurements only have to be once for all CUDA GPUs.

STARPU_PERF_MODEL_HOMOGENEOUS_OPENCL When set to 1, StarPU will assume that all OpenCL devices have the same performance, and thus share performance models for them, thus allowing kernel calibration to be much faster, since measurements only have to be once for all OpenCL GPUs.

STARPU_PERF_MODEL_HOMOGENEOUS_MPI_MS When set to 1, StarPU will assume that all MPI Slave devices have the same performance, and thus share performance models for them, thus allowing kernel calibration to be much faster, since measurements only have to be once for all MPI Slaves.

STARPU_HOSTNAME When set, force the hostname to be used when managing performance model files. Models are indexed by machine name. When running for example on a homogenenous cluster, it is possible to share the models between machines by setting `export STARPU_HOSTNAME=some_global_name`.

STARPU_MPI_HOSTNAMES Similar to STARPU_HOSTNAME but to define multiple nodes on a heterogeneous cluster. The variable is a list of hostnames that will be assigned to each StarPU-MPI rank considering their position and the value of starpu_mpi_world_rank() on each rank. When running, for example, on a heterogeneous cluster, it is possible to set individual models for each machine by setting `export STARPU_MPI←_HOSTNAMES="name0 name1 name2"`. Where rank 0 will receive `name0`, rank1 will receive `name1`, and so on. This variable has precedence over STARPU_HOSTNAME.

STARPU_OPENCL_PROGRAM_DIR Specify the directory where the OpenCL codelet source files are located. The function starpu_opencl_load_program_source() looks for the codelet in the current directory, in the directory specified by the environment variable STARPU_OPENCL_PROGRAM_DIR, in the directory `share/starpu/opencl` of the installation directory of StarPU, and finally in the source directory of StarPU.

STARPU_SILENT Disable verbose mode at runtime when StarPU has been configured with the option --enable-verbose. Also disable the display of StarPU information and warning messages.

STARPU_MPI_DEBUG_LEVEL_MIN Set the minimum level of debug when StarPU has been configured with the option --enable-mpi-verbose.

STARPU_MPI_DEBUG_LEVEL_MAX Set the maximum level of debug when StarPU has been configured with the option --enable-mpi-verbose.

STARPU_LOGFILENAME Specify in which file the debugging output should be saved to.

STARPU_FXT_PREFIX Specify in which directory to save the generated trace if FxT is enabled.

STARPU_FXT_SUFFIX Specify in which file to save the generated trace if FxT is enabled.

STARPU_FXT_TRACE Enable (1) or disable (0) the FxT trace generation in `/tmp/prof_file_XXX_YYY` (the directory and file name can be changed with STARPU_FXT_PREFIX and STARPU_FXT_SUFFIX). Default value is Disable.

STARPU_FXT_EVENTS Specify which events will be recorded in traces. By default, all events (but `VERBOSE_EXTRA` ones) are recorded. One can set this variable to a comma- or pipe-separated list of the following categories, to record only events belonging to the selected categories:

- `USER`
- `TASK`
- `TASK_VERBOSE`
- `TASK_VERBOSE_EXTRA`
- `DATA`
- `DATA_VERBOSE`
- `WORKER`
- `WORKER_VERBOSE`
- `DSM`
- `DSM_VERBOSE`
- `SCHED`
- `SCHED_VERBOSE`
- `LOCK`
- `LOCK_VERBOSE`
- `EVENT`
- `EVENT_VERBOSE`
- `MPI`
- `MPI_VERBOSE`
- `MPI_VERBOSE_EXTRA`
- `HYP`
- `HYP_VERBOSE`

The choice of which categories have to be recorded is a tradeoff between required information for offline analyzis and acceptable overhead introduced by tracing. For instance, to inspect with ViTE which tasks workers execute, one has to at least select the `TASK` category.

Events in `VERBOSE_EXTRA` are very costly to record and can have an important impact on application performances. This is why there are disabled by default, and one has to explicitly select their categories using this variable to record them.

STARPU_LIMIT_CUDA_devid_MEM Specify the maximum number of megabytes that should be available to the application on the CUDA device with the identifier `devid`. This variable is intended to be used for experimental purposes as it emulates devices that have a limited amount of memory. When defined, the variable overwrites the value of the variable STARPU_LIMIT_CUDA_MEM.

STARPU_LIMIT_CUDA_MEM Specify the maximum number of megabytes that should be available to the application on each CUDA devices. This variable is intended to be used for experimental purposes as it emulates devices that have a limited amount of memory.

STARPU_LIMIT_OPENCL_devid_MEM Specify the maximum number of megabytes that should be available to the application on the OpenCL device with the identifier `devid`. This variable is intended to be used for experimental purposes as it emulates devices that have a limited amount of memory. When defined, the variable overwrites the value of the variable STARPU_LIMIT_OPENCL_MEM.

STARPU_LIMIT_OPENCL_MEM Specify the maximum number of megabytes that should be available to the application on each OpenCL devices. This variable is intended to be used for experimental purposes as it emulates devices that have a limited amount of memory.

STARPU_LIMIT_HIP_devid_MEM Specify the maximum number of megabytes that should be available to the application on the HIP device with the identifier `devid`. This variable is intended to be used for experimental purposes as it emulates devices that have a limited amount of memory. When defined, the variable overwrites the value of the variable STARPU_LIMIT_HIP_MEM.

STARPU_LIMIT_HIP_MEM Specify the maximum number of megabytes that should be available to the application on each HIP devices. This variable is intended to be used for experimental purposes as it emulates devices that have a limited amount of memory.

STARPU_LIMIT_CPU_MEM Specify the maximum number of megabytes that should be available to the application in the main CPU memory. Setting it enables allocation cache in main memory. Setting it to zero lets StarPU overflow memory.

Note: for now not all StarPU allocations get throttled by this parameter. Notably MPI reception are not throttled unless STARPU_MPI_MEM_THROTTLE is set to 1.

STARPU_LIMIT_CPU_NUMA_devid_MEM Specify the maximum number of megabytes that should be available to the application on the NUMA node with the OS identifier `devid`. Setting it overrides the value of STARPU_LIMIT_CPU_MEM.

STARPU_LIMIT_CPU_NUMA_MEM Specify the maximum number of megabytes that should be available to the application on each NUMA node. This is the same as specifying that same amount with STARPU_LIMIT_CPU_NUMA_devid_MEM for each NUMA node number. The total memory available to StarPU will thus be this amount multiplied by the number of NUMA nodes used by StarPU. Any STARPU_LIMIT_CPU_NUMA_devid_MEM additionally specified will take over STARPU_LIMIT_CPU_NUMA_MEM.

STARPU_LIMIT_BANDWIDTH Specify the maximum available PCI bandwidth of the system in MB/s. This can only be effective with simgrid simulation. This allows to easily override the bandwidths stored in the platform file generated from measurements on the native system. This can thus be used accelerate or slow down the system bandwidth.

STARPU_SUBALLOCATOR Enable (1) or disable (0) the StarPU suballocator. Default value is to enable it to amortize the cost of GPU and pinned RAM allocations for small allocations: StarPU allocate large chunks of memory at a time, and suballocates the small buffers within them.

STARPU_MINIMUM_AVAILABLE_MEM Specify the minimum percentage of memory that should be available in GPUs, i.e. not used at all by StarPU (or in main memory, when using out of core), below which a eviction pass is performed. Default value is 0%.

STARPU_TARGET_AVAILABLE_MEM Specify the target percentage of memory that should be available in GPUs, i.e. not used at all by StarPU (or in main memory, when using out of core), when performing a periodic eviction pass. Default value is 0%.

STARPU_MINIMUM_CLEAN_BUFFERS Specify the minimum percentage of number of buffers that should be clean in GPUs (or in main memory, when using out of core), i.e. used by StarPU, but for which a copy is available in memory (or on disk, when using out of core), below which asynchronous writebacks will be issued. Default value is 5%.

STARPU_TARGET_CLEAN_BUFFERS Specify the target percentage of number of buffers that should be reached in GPUs (or in main memory, when using out of core), i.e. used by StarPU, but for which a copy is available in memory (or on disk, when using out of core), when performing an asynchronous writeback pass. Default value is 10%.

STARPU_DISK_SWAP Specify a path where StarPU can push data when the main memory is getting full.

STARPU_DISK_SWAP_BACKEND Specify the backend to be used by StarPU to push data when the main memory is getting full. Default value is `unistd` (i.e. using read/write functions), other values are `stdio` (i.e. using fread/fwrite), `unistd_o_direct` (i.e. using read/write with O_DIRECT), `leveldb` (i.e. using a leveldb database), and `hdf5` (i.e. using HDF5 library).

STARPU_DISK_SWAP_SIZE Specify the maximum size in MiB to be used by StarPU to push data when the main memory is getting full. Default value is unlimited.

STARPU_LIMIT_MAX_SUBMITTED_TASKS Allow users to control the task submission flow by specifying to StarPU a maximum number of submitted tasks allowed at a given time, i.e. when this limit is reached task submission becomes blocking until enough tasks have completed, specified by STARPU_LIMIT_MIN_SUBMITTED_TASKS. Setting it enables allocation cache buffer reuse in main memory. See How To Reduce The Memory Footprint Of Internal Data Structures.

STARPU_LIMIT_MIN_SUBMITTED_TASKS Allow users to control the task submission flow by specifying to StarPU a submitted task threshold to wait before unblocking task submission. This variable has to be used in conjunction with STARPU_LIMIT_MAX_SUBMITTED_TASKS which puts the task submission thread to sleep. Setting it enables allocation cache buffer reuse in main memory. See How To Reduce The Memory Footprint Of Internal Data Structures.

STARPU_TRACE_BUFFER_SIZE Set the buffer size for recording trace events in MiB. Setting it to a big size allows to avoid pauses in the trace while it is recorded on the disk. This however also consumes memory, of course. Default value is 64.

STARPU_GENERATE_TRACE When set to `1`, indicate that StarPU should automatically generate a Paje trace when starpu_shutdown() is called.

STARPU_GENERATE_TRACE_OPTIONS When the variable STARPU_GENERATE_TRACE is set to `1` to generate a Paje trace, this variable can be set to specify options (see `starpu_fxt_tool -help`).

STARPU_ENABLE_STATS Enable gathering various data statistics (Data Statistics).

STARPU_MEMORY_STATS When set to 0, disable the display of memory statistics on data which have not been unregistered at the end of the execution (Memory Feedback).

STARPU_MAX_MEMORY_USE When set to 1, display at the end of the execution the maximum memory used by StarPU for internal data structures during execution.

STARPU_BUS_STATS Enable the display of data transfers statistics when calling starpu_shutdown() (Profiling). By default, statistics are printed on the standard error stream, use the environment variable STARPU_BUS_STATS_FILE to define another filename.

STARPU_BUS_STATS_FILE Define the name of the file where to display data transfers statistics, see STARPU_BUS_STATS.

STARPU_WORKER_STATS Enable the display of workers statistics when calling starpu_shutdown() (Profiling). When combined with the environment variable STARPU_PROFILING, it displays the energy consumption (Energy-based Scheduling). By default, statistics are printed on the standard error stream, use the environment variable STARPU_WORKER_STATS_FILE to define another filename.

STARPU_WORKER_STATS_FILE Define the name of the file where to display workers statistics, see STARPU_WORKER_STATS.

STARPU_STATS When set to 0, data statistics will not be displayed at the end of the execution of an application (Data Statistics).

STARPU_WATCHDOG_TIMEOUT When set to a value other than 0, allows to make StarPU print an error message whenever StarPU does not terminate any task for the given time (in µs), but lets the application continue normally. Should be used in combination with STARPU_WATCHDOG_CRASH (see Detecting Stuck Conditions).

STARPU_WATCHDOG_CRASH When set to a value other than 0, trigger a crash when the watch dog is reached, thus allowing to catch the situation in gdb, etc (see Detecting Stuck Conditions)

STARPU_WATCHDOG_DELAY Delay the activation of the watchdog by the given time (in µs). This can be convenient for letting the application initialize data etc. before starting to look for idle time.

STARPU_TASK_PROGRESS Print the progression of tasks. This is convenient to determine whether a program is making progress in task execution, or is just stuck.

STARPU_TASK_BREAK_ON_PUSH When this variable contains a job id, StarPU will raise `SIGTRAP` when the task with that job id is being pushed to the scheduler, which will be nicely caught by debuggers (see Debugging Scheduling)

STARPU_TASK_BREAK_ON_SCHED When this variable contains a job id, StarPU will raise `SIGTRAP` when the task with that job id is being scheduled by the scheduler (at a scheduler-specific point), which will be nicely caught by debuggers. This only works for schedulers which have such a scheduling point defined (see Debugging Scheduling)

STARPU_TASK_BREAK_ON_POP When this variable contains a job id, StarPU will raise `SIGTRAP` when the task with that job id is being popped from the scheduler, which will be nicely caught by debuggers (see Debugging Scheduling)

STARPU_TASK_BREAK_ON_EXEC When this variable contains a job id, StarPU will raise `SIGTRAP` when the task with that job id is being executed, which will be nicely caught by debuggers (see Debugging Scheduling)

STARPU_DISABLE_KERNELS When set to a value other than 1, it disables actually calling the kernel functions, thus allowing to quickly check that the task scheme is working properly, without performing the actual application-provided computation.

STARPU_HISTORY_MAX_ERROR History-based performance models will drop measurements which are really far froom the measured average. This specifies the allowed variation. Default value is 50 (%), i.e. the measurement is allowed to be x1.5 faster or /1.5 slower than the average.

STARPU_RAND_SEED The random scheduler and some examples use random numbers for their own working. Depending on the examples, the seed is by default juste always 0 or the current time() (unless SimGrid mode is enabled, in which case it is always 0). STARPU_RAND_SEED allows to set the seed to a specific value.

STARPU_GLOBAL_ARBITER When set to a positive value, StarPU will create a arbiter, which implements an advanced but centralized management of concurrent data accesses (see Concurrent Data Accesses).

STARPU_USE_NUMA When defined to 1, NUMA nodes are taking into account by StarPU, i.e. StarPU will expose one StarPU memory node per NUMA node, and will thus schedule tasks according to data locality, migrated data when appropriate, etc.

STARPU_MAIN_RAM is then associated to the NUMA node associated to the first CPU worker if it exists, the NUMA node associated to the first GPU discovered otherwise. If StarPU doesn't find any NUMA node after these steps, STARPU_MAIN_RAM is the first NUMA node discovered by StarPU.

Applications should thus rather pass a `NULL` pointer and a -1 memory node to `starpu_data_*_register` functions, so that StarPU can manage memory as it wishes.

If the application wants to control memory allocation on NUMA nodes for some data, it can use starpu_malloc_on_node and pass the memory node to the `starpu_data_*_register` functions to tell StarPU where the allocation was made. starpu_memory_nodes_get_count_by_kind() and starpu_memory_node_get_ids_by_type() can be used to get the memory nodes numbers of the CPU memory nodes.

starpu_memory_nodes_numa_id_to_devid() and starpu_memory_nodes_numa_devid_to_id() are also available to convert between OS NUMA id and StarPU memory node number.

If this variable is unset, or set to 0, CPU memory is considered as only one memory node (STARPU_MAIN_RAM) and it will be up to the OS to manage migration etc. and the StarPU scheduler will not know about it.

STARPU_IDLE_FILE When defined, a file named after its contents will be created at the end of the execution. This file will contain the sum of the idle times of all the workers.

STARPU_HWLOC_INPUT When defined to the path of an XML file, `hwloc` will use this file as input instead of detecting the current platform topology, which can save significant initialization time.

To produce this XML file, use `lstopo file.xml`

STARPU_CATCH_SIGNALS By default, StarPU catch signals `SIGINT`, `SIGSEGV` and `SIGTRAP` to perform final actions such as dumping FxT trace files even though the application has crashed. Setting this variable to a value other than 1 will disable this behaviour. This should be done on JVM systems which may use these signals for their own needs. The flag can also be set through the field starpu_conf::catch_signals.

STARPU_DISPLAY_BINDINGS Display the binding of all processes and threads running on the machine. If MPI is enabled, display the binding of each node.
Users can manually display the binding by calling starpu_display_bindings().

7.6 Configuring The Hypervisor

SC_HYPERVISOR_POLICY Choose between the different resizing policies proposed by StarPU for the hypervisor: `idle`, `app_driven`, `feft_lp`, `teft_lp`, `ispeed_lp`, `throughput_lp` etc.

Use `SC_HYPERVISOR_POLICY=help` to get the list of available policies for the hypervisor

SC_HYPERVISOR_TRIGGER_RESIZE Choose how should the hypervisor be triggered: `speed` if the resizing algorithm should be called whenever the speed of the context does not correspond to an optimal precomputed value, `idle` it the resizing algorithm should be called whenever the workers are idle for a period longer than the value indicated when configuring the hypervisor.

SC_HYPERVISOR_START_RESIZE Indicate the moment when the resizing should be available. The value correspond to the percentage of the total time of execution of the application. Default value is the resizing frame.

SC_HYPERVISOR_MAX_SPEED_GAP Indicate the ratio of speed difference between contexts that should trigger the hypervisor. This situation may occur only when a theoretical speed could not be computed and the hypervisor has no value to compare the speed to. Otherwise the resizing of a context is not influenced by the the speed of the other contexts, but only by the the value that a context should have.

SC_HYPERVISOR_STOP_PRINT By default the values of the speed of the workers is printed during the execution of the application. If the value 1 is given to this environment variable this printing is not done.

SC_HYPERVISOR_LAZY_RESIZE By default the hypervisor resizes the contexts in a lazy way, that is workers are firstly added to a new context before removing them from the previous one. Once this workers are clearly taken into account into the new context (a task was popped there) we remove them from the previous one. However if the application would like that the change in the distribution of workers should change right away this variable should be set to 0

SC_HYPERVISOR_SAMPLE_CRITERIA By default the hypervisor uses a sample of flops when computing the speed of the contexts and of the workers. If this variable is set to `time` the hypervisor uses a sample of time (10% of an approximation of the total execution time of the application)

Chapter 8

Configuration and initialization

This section explains the relationship between configure options, compilation options and environment variables used by StarPU.

1. Configure options are used during the installation process to enable or disable specific features and libraries. These options are set using flags like --enable-maxcpus, which can be used to set the maximum number of CPUs that can be used by StarPU.
2. Compilation options are used to set specific parameters during the compilation process, such as the optimization level, architecture type, and debugging options.
3. Environment variables are used to set runtime parameters and control the behavior of the StarPU library. For example, the STARPU_NCPUS environment variable can be used to specify the number of CPUs to use at runtime, overriding the value set during compilation or installation.

Options can also be set with the different fields of the starpu_conf parameter given to starpu_init(), such as starpu_conf::ncpus, which is used to specify the number of CPUs that StarPU should use for computations.

Part II

StarPU Basics

Chapter 9

Organization

This part presents the basic knowledge of StarPU. It should be read to understand how StarPU works and how to execute a basic StarPU application.

- Chapter StarPU Applications, setting up Your Own Code shows how to create and run your own StarPU applications.
- Chapter Basic Examples shows how to implement simple programs that submit tasks to StarPU.
- Chapter Full source code for the 'Scaling a Vector' example gives the full source code for a vector scaling application.

The next chapters cover the most important and core concepts in StarPU:

- Chapter Tasks In StarPU explains the basic information on tasks management.
- Chapter Data Management shows how to manage the data layout of your application data by using the different data interfaces provided by StarPU.
- Chapter Scheduling explains the scheduling policies provided by StarPU.

Some examples applications are provided from the StarPU sources for you to try. Chapter Examples in StarPU Sources lists these applications.

Chapter 10

StarPU Applications

10.1 Setting Flags for Compiling, Linking and Running Applications

StarPU provides a `pkg-config` executable to facilitate the retrieval of necessary compiler and linker flags. This is useful when compiling and linking an application with StarPU, as certain flags or libraries (such as `CUDA` or `libspe2`) may be required.
If StarPU is not installed in a standard location, the path of StarPU's library must be specified in the environment variable `PKG_CONFIG_PATH` to allow `pkg-config` to find it. For example, if StarPU is installed in `$STARPU_PATH`, you can set the variable `PKG_CONFIG_PATH` like this:

```
$ export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$STARPU_PATH/lib/pkgconfig
```

The flags required to compile or link against StarPU are then accessible with the following commands:

```
$ pkg-config --cflags starpu-1.4  # options for the compiler
$ pkg-config --libs starpu-1.4    # options for the linker
```

Please note that it is still possible to use the API provided in StarPU version 1.0 by calling `pkg-config` with the `starpu-1.0` package. Similar packages are provided for `starpumpi-1.0` and `starpufft-1.0`. For the API provided in StarPU version 0.9, you can use `pkg-config` with the `libstarpu` package. Similar packages are provided for `libstarpumpi` and `libstarpufft`.
Make sure that `pkg-config -libs starpu-1.4` produces valid output before going further. To achieve this, make sure that your `PKG_CONFIG_PATH` is correctly set to the location where `starpu-1.4.pc` was installed during the `make install` process.
Furthermore, if you intend to link your application statically, remember to include the `-static` option during the linking process.
Additionally, for runtime execution, it is necessary to set the `LD_LIBRARY_PATH` environment variable. This ensures that dynamic libraries are located and loaded correctly during runtime.

```
$ export LD_LIBRARY_PATH=$STARPU_PATH/lib:$LD_LIBRARY_PATH
```

And finally you should set the `PATH` variable to get access to various StarPU tools:

```
$ export PATH=$PATH:$STARPU_PATH/bin
```

Run the following command to ensure that StarPU is executing properly and successfully detecting your hardware. If any issues arise, examine the output of `lstopo` from the `hwloc` project and report any problems either to the hwloc project or to us.

```
$ starpu_machine_display
```

A tool is provided to help set all the environment variables needed by StarPU. Once StarPU is installed in a specific directory, calling the script `bin/starpu_env` will set in your current environment the variables `STARPU_PATH`, `LD_LIBRARY_PATH`, `PKG_CONFIG_PATH`, `PATH` and `MANPATH`.

```
$ source $STARPU_PATH/bin/starpu_env
```

10.2 Integrating StarPU in a Build System

10.2.1 Integrating StarPU in a Make Build System

When using a Makefile, the following lines can be added to set the options for the compiler and the linker:

```
CFLAGS          +=      $$(pkg-config --cflags starpu-1.4)
LDLIBS          +=      $$(pkg-config --libs starpu-1.4)
```

If you have a `test-starpu.c` file containing for instance:

```
#include <starpu.h>
#include <stdio.h>
int main(void)
{
    int ret;
    ret = starpu_init(NULL);
    if (ret != 0)
    {
        return 1;
    }
    printf("%d CPU cores\n", starpu_worker_get_count_by_type(STARPU_CPU_WORKER));
    printf("%d CUDA GPUs\n", starpu_worker_get_count_by_type(STARPU_CUDA_WORKER));
    printf("%d OpenCL GPUs\n", starpu_worker_get_count_by_type(STARPU_OPENCL_WORKER));
    starpu_shutdown();
    return 0;
}
```

You can build it with `make test-starpu` and run it with `./test-starpu`

10.2.2 Integrating StarPU in a CMake Build System

This section shows a minimal example integrating StarPU in an existing application's CMake build system.

Let's assume we want to build an executable from the following source code using CMake:

```
#include <starpu.h>
#include <stdio.h>
int main(void)
{
    int ret;
    ret = starpu_init(NULL);
    if (ret != 0)
    {
        return 1;
    }
    printf("%d CPU cores\n", starpu_worker_get_count_by_type(STARPU_CPU_WORKER));
    printf("%d CUDA GPUs\n", starpu_worker_get_count_by_type(STARPU_CUDA_WORKER));
    printf("%d OpenCL GPUs\n", starpu_worker_get_count_by_type(STARPU_OPENCL_WORKER));
    starpu_shutdown();
    return 0;
}
```

The `CMakeLists.txt` file below uses the Pkg-Config support from CMake to autodetect the StarPU installation and library dependences (such as `libhwloc`) provided that the `PKG_CONFIG_PATH` variable is set, and is sufficient to build a statically-linked executable. This example has been successfully tested with CMake 3.2, though it may work with earlier CMake 3.x versions.

```
{File CMakeLists.txt}
cmake_minimum_required (VERSION 3.2)
project (hello_starpu)
find_package(PkgConfig)
pkg_check_modules(STARPU REQUIRED starpu-1.4)
if (STARPU_FOUND)
    include_directories (${STARPU_INCLUDE_DIRS})
    link_directories    (${STARPU_STATIC_LIBRARY_DIRS})
    link_libraries      (${STARPU_STATIC_LIBRARIES})
else (STARPU_FOUND)
    message(FATAL_ERROR "StarPU not found")
endif()
add_executable(hello_starpu hello_starpu.c)
```

The following `CMakeLists.txt` implements an alternative, more complex strategy, still relying on Pkg-Config, but also taking into account additional flags. While more complete, this approach makes CMake's build types (Debug, Release, ...) unavailable because of the direct affectation to variable `CMAKE_C_FLAGS`. If both the full flags support and the build types support are needed, the `CMakeLists.txt` below may be altered to work with `CMAKE_C_FLAGS_RELEASE`, `CMAKE_C_FLAGS_DEBUG`, and others as needed. This example has been successfully tested with CMake 3.2, though it may work with earlier CMake 3.x versions.

```
{File CMakeLists.txt}
cmake_minimum_required (VERSION 3.2)
project (hello_starpu)
find_package(PkgConfig)
pkg_check_modules(STARPU REQUIRED starpu-1.4)
# This section must appear before ’add_executable’
```

```
if (STARPU_FOUND)
# CFLAGS other than -I
    foreach(CFLAG ${STARPU_CFLAGS_OTHER})
        set (CMAKE_C_FLAGS "${CMAKE_C_FLAGS} ${CFLAG}")
    endforeach()
    # Static LDFLAGS other than -L
    foreach(LDFLAG ${STARPU_STATIC_LDFLAGS_OTHER})
        set (CMAKE_EXE_LINKER_FLAGS "${CMAKE_EXE_LINKER_FLAGS} ${LDFLAG}")
    endforeach()
    # -L directories
    link_directories(${STARPU_STATIC_LIBRARY_DIRS})
else (STARPU_FOUND)
    message(FATAL_ERROR "StarPU not found")
endif()
add_executable(hello_starpu hello_starpu.c)
# This section must appear after 'add_executable'
if (STARPU_FOUND)
# -I directories
    target_include_directories(hello_starpu PRIVATE ${STARPU_INCLUDE_DIRS})
    # Static -l libs
    target_link_libraries(hello_starpu PRIVATE ${STARPU_STATIC_LIBRARIES})
endif()
```

10.3 Running a Basic StarPU Application

Basic examples using StarPU are built in the directory `examples/basic_examples/` (and installed in `$STARPU_PATH/lib/starpu/examples/`). You can for example run the example `vector_scal`.

```
$ ./examples/basic_examples/vector_scal
BEFORE: First element was 1.000000
AFTER: First element is 3.140000
```

When StarPU is used for the first time, the directory `$STARPU_HOME/.starpu/` is created, performance models will be stored in this directory (STARPU_HOME).

Please note that buses are benchmarked when StarPU is launched for the first time. This may take a few minutes, or less if `libhwloc` is installed. This step is done only once per user and per machine.

10.4 Running a Basic StarPU Application on Microsoft Visual C

Batch files are provided to run StarPU applications under Microsoft Visual C. They are installed in `$STARPU_PATH/bin/msvc`.

To execute a StarPU application, you first need to set the environment variable STARPU_PATH.

```
c:\....> cd c:\cygwin\home\ci\starpu\
c:\....> set STARPU_PATH=c:\cygwin\home\ci\starpu\
c:\....> cd bin\msvc
c:\....> starpu_open.bat starpu_simple.c
```

The batch script will run Microsoft Visual C with a basic project file to run the given application.

The batch script `starpu_clean.bat` can be used to delete all compilation generated files.

The batch script `starpu_exec.bat` can be used to compile and execute a StarPU application from the command prompt.

```
c:\....> cd c:\cygwin\home\ci\starpu\
c:\....> set STARPU_PATH=c:\cygwin\home\ci\starpu\
c:\....> cd bin\msvc
c:\....> starpu_exec.bat ..\..\..\..\examples\basic_examples\hello_world.c

MSVC StarPU Execution
...
/out:hello_world.exe
...
Hello world (params = {1, 2.00000})
Callback function got argument 0000042
c:\....>
```

10.5 Kernel Threads Started by StarPU

StarPU automatically binds one thread per CPU core. It does not use SMT/hyperthreading because kernels are usually already optimized for using a full core, and using hyperthreading would make kernel calibration rather random.
Since driving GPUs is a CPU-consuming task, StarPU dedicates one core per GPU.
While StarPU tasks are executing, the application is not supposed to do computations in the threads it starts itself, tasks should be used instead.
If the application needs to reserve some cores for its own computations, it can do so with the field starpu_conf::reserve_ncpus, get the core IDs with starpu_get_next_bindid(), and bind to them with starpu_bind_thread_on().
Another option is for the application to pause StarPU by calling starpu_pause(), then to perform its own computations, and then to resume StarPU by calling starpu_resume() so that StarPU can execute tasks.
If a computation library used by the application actually creates its own thread, it may be useful to call starpu_bind_thread_on_worker() before e.g. initializing the library, so that the library records which binding it is supposed to use. And then call starpu_bind_thread_on_main() again, or starpu_bind_thread_on_cpu() if a core was reserved with starpu_get_next_bindid().
In case that computation library wants to bind threads itself, and uses physical numbering instead of logical numbering (as defined by hwloc), starpu_cpu_os_index() can be used to convert from StarPU cpuid to OS cpu index.

10.6 Enabling OpenCL

When both CUDA and OpenCL drivers are enabled, StarPU will launch an OpenCL worker for NVIDIA GPUs only if CUDA is not already running on them. This design choice was necessary as OpenCL and CUDA can not run at the same time on the same NVIDIA GPU, as there is currently no interoperability between them.
To enable OpenCL, you need either to disable CUDA when configuring StarPU:

```
$ ./configure --disable-cuda
```

or when running applications:

```
$ STARPU_NCUDA=0 ./application
```

OpenCL will automatically be started on any device not yet used by CUDA. So on a machine running 4 GPUS, it is therefore possible to enable CUDA on 2 devices, and OpenCL on the other 2 devices by calling:

```
$ STARPU_NCUDA=2 ./application
```

10.7 Storing Performance Model Files

StarPU stores performance model files for bus benchmarking and codelet profiles in different directories.
By default, all files are stored in `$STARPU_HOME/.starpu/sampling`.
If the environment variable STARPU_HOME is not defined, its default value is `$HOME` on Unix environments, and `$USERPROFILE` on Windows environments.
Environment variables STARPU_PERF_MODEL_DIR and STARPU_PERF_MODEL_PATH can also be used to specify other directories in which to store performance files (Simulated Benchmarks).
The configure option --with-perf-model-dir can also be used to define a performance model directory.
When looking for performance files either for bus benchmarking or for codelet performances, StarPU

- first looks in the directory specified by the environment variable STARPU_PERF_MODEL_DIR
- then looks in the directory specified by the configure option --with-perf-model-dir
  or in $STARPU_HOME/.starpu/sampling if the option is not set
- then looks in the directories specified by the environment variable STARPU_PERF_MODEL_PATH
- and finally looks in `$prefix/share/starpu/perfmodels/sampling`

If the files are not present and must be created, they will be created in the first defined directory from the list above.

```
rm -rf $PWD/xxx && STARPU_PERF_MODEL_DIR=$PWD/xxx ./application
```

will use performance model files from the directory `$STARPU_HOME/.starpu/sampling` if they are available, otherwise will create these files in `$STARPU_PERF_MODEL_DIR`.
To know the list of directories StarPU will search for performances files, one can use the tool `starpu_perfmodel_display`

```
$ starpu_perfmodel_display -d
directory: </home/user1/.starpu/sampling/codelets/45/>
directory: </usr/local/install/share/starpu/perfmodels/sampling/codelets/45/>

$ STARPU_PERF_MODEL_DIR=/tmp/xxx starpu_perfmodel_display -d
directory: </tmp/xxx/codelets/45/>
directory: </home/user1/.starpu/sampling/codelets/45/>
directory: </usr/local/install/share/starpu/perfmodels/sampling/codelets/45/>
```

When using the variable STARPU_PERF_MODEL_DIR, the directory will be created if it does not exist when dumping new performance model files.
When using the variable STARPU_PERF_MODEL_PATH, only existing directories will be taken into account.

```
$ mkdir /tmp/yyy && STARPU_PERF_MODEL_DIR=/tmp/xxx STARPU_PERF_MODEL_PATH=/tmp/zzz:/tmp/yyy starpu_perfmodel_
[starpu][adrets][_perf_model_add_dir] Warning: directory </tmp/zzz> as set by variable STARPU_PERF_MODEL_PATH
directory: </tmp/xxx/codelets/45/>
directory: </home/user1/.starpu/sampling/codelets/45/>
directory: </tmp/yyy/codelets/45/>
directory: </usr/local/install/share/starpu/perfmodels/sampling/codelets/45/>
```

Once your application has created the performance files in a given directory, it is thus possible to move these files in another location and keep using them.

```
./application
# files are created in $HOME/.starpu/sampling
mv $HOME/.starpu/sampling /usr/local/starpu/sampling
STARPU_PERF_MODEL_DIR=/usr/local/starpu/sampling ./application
```

Chapter 11

Basic Examples

11.1 Hello World

This section shows how to implement a simple program that submits a task to StarPU. The full source code for this example is available in the file `examples/basic_examples/hello_world.c`

11.1.1 Required Headers

The header `starpu.h` should be included in any code using StarPU.

```
#include <starpu.h>
```

11.1.2 Defining A Codelet

A codelet is a structure that represents a computational kernel. Such a codelet may contain an implementation of the same kernel on different architectures (e.g. CUDA, x86, ...). For compatibility, make sure that the whole structure is properly initialized to zero, either by using the function starpu_codelet_init(), or by letting the compiler implicitly do it as examplified below.

The field starpu_codelet::nbuffers specifies the number of data buffers that are manipulated by the codelet. Here, the codelet does not access or modify any data that is controlled by our data management library.

We create a codelet which may only be executed on CPUs. When a CPU core will execute a codelet, it will call the function `cpu_func`, which *must* have the following prototype:

```
void cpu_func(void *buffers[], void *cl_arg);
```

In this example, we can ignore the first argument of this function which gives a description of the input and output buffers (e.g. the size and the location of the matrices) since there is none. We also ignore the second argument, which is a pointer to optional arguments for the codelet.

```
void cpu_func(void *buffers[], void *cl_arg)
{
    printf("Hello world\n");
}
struct starpu_codelet cl =
{
    .cpu_funcs = { cpu_func },
    .nbuffers = 0
};
```

11.1.3 Submitting A Task

Before submitting any tasks to StarPU, starpu_init() must be called, or starpu_initialize() must be called by giving application arguments. The `NULL` argument specifies that we use the default configuration. Tasks can then be submitted until the termination of StarPU – done by a call to starpu_shutdown().

In the example below, a task structure is allocated by a call to starpu_task_create(). This function allocates and fills the task structure with its default settings, it does not submit the task to StarPU.

The field starpu_task::cl is a pointer to the codelet which the task will execute: in other words, the codelet structure describes which computational kernel should be offloaded on the different architectures, and the task structure is a wrapper containing a codelet and the piece of data on which the codelet should operate.

If the field starpu_task::synchronous is non-zero, task submission will be synchronous: the function starpu_task_submit() will not return until the task has been executed. Note that the function starpu_shutdown() does not guarantee that asynchronous tasks have been executed before it returns, starpu_task_wait_for_all() can be used to this effect, or data can be unregistered (starpu_data_unregister()), which will implicitly wait for all the tasks scheduled to work on it, unless explicitly disabled thanks to starpu_data_set_default_sequential_consistency_flag() or starpu_data_set_sequential_consistency_flag().

```
int main(int argc, char **argv)
{
    /* initialize StarPU */
    starpu_init(NULL);
    struct starpu_task *task = starpu_task_create();
    task->cl = &cl; /* Pointer to the codelet defined above */
    /* starpu_task_submit will be a blocking call.  If unset,
starpu_task_wait() needs to be called after submitting the task.  */
    task->synchronous = 1;
    /* submit the task to StarPU */
    starpu_task_submit(task);
    /* terminate StarPU */
    starpu_shutdown();
    return 0;
}
```

11.1.4 Execution Of Hello World

```
$ make hello_world
cc $(pkg-config --cflags starpu-1.4) hello_world.c -o hello_world $(pkg-config --libs starpu-1.4)
$ ./hello_world
Hello world
```

11.1.5 Passing Arguments To The Codelet

The optional field starpu_task::cl_arg field is a pointer to a buffer (of size starpu_task::cl_arg_size) with some parameters for the kernel described by the codelet. For instance, if a codelet implements a computational kernel that multiplies its input vector by a constant, the constant could be specified by the means of this buffer, instead of registering it as a StarPU data. It must however be noted that StarPU avoids making copy whenever possible and rather passes the pointer as such, so the buffer which is pointed to must be kept allocated until the task terminates, and if several tasks are submitted with various parameters, each of them must be given a pointer to their own buffer.

```
struct params
{
    int i;
    float f;
};
void cpu_func(void *buffers[], void *cl_arg)
{
    struct params *params = cl_arg;
    printf("Hello world (params = {%i, %f} )\n", params->i, params->f);
}
```

As said before, the field starpu_codelet::nbuffers specifies the number of data buffers which are manipulated by the codelet. It does not count the argument — the parameter `cl_arg` of the function `cpu_func` — since it is not managed by our data management library, but just contains trivial parameters.

Be aware that this may be a pointer to a *copy* of the actual buffer, and not the pointer given by the programmer: if the codelet modifies this buffer, there is no guarantee that the initial buffer will be modified as well: this for instance implies that the buffer cannot be used as a synchronization medium. If synchronization is needed, data has to be registered to StarPU, see Vector Scaling.

```
int main(int argc, char **argv)
{
    /* initialize StarPU */
    starpu_init(NULL);
    struct starpu_task *task = starpu_task_create();
    task->cl = &cl; /* Pointer to the codelet defined above */
    struct params params = { 1, 2.0f };
    task->cl_arg = &params;
    task->cl_arg_size = sizeof(params);
    /* starpu_task_submit will be a blocking call */
    task->synchronous = 1;
    /* submit the task to StarPU */
    starpu_task_submit(task);
    /* terminate StarPU */
    starpu_shutdown();
    return 0;
}
```

```
$ make hello_world
cc $(pkg-config --cflags starpu-1.4) hello_world.c -o hello_world $(pkg-config --libs starpu-1.4)
$ ./hello_world
Hello world (params = {1, 2.000000} )
```

11.1.6 Defining A Callback

Once a task has been executed, an optional callback function starpu_task::callback_func is called when defined. While the computational kernel could be offloaded on various architectures, the callback function is always executed on a CPU. The pointer starpu_task::callback_arg is passed as an argument to the callback function. The prototype of a callback function must be:

```
void callback_function(void *);
void callback_func(void *callback_arg)
{
    printf("Callback function (arg %x)\n", callback_arg);
}
int main(int argc, char **argv)
{
    /* initialize StarPU */
    starpu_init(NULL);
    struct starpu_task *task = starpu_task_create();
    task->cl = &cl; /* Pointer to the codelet defined above */
    task->callback_func = callback_func;
    task->callback_arg = 0x42;
    /* starpu_task_submit will be a blocking call */
    task->synchronous = 1;
    /* submit the task to StarPU */
    starpu_task_submit(task);
    /* terminate StarPU */
    starpu_shutdown();
    return 0;
}
```

```
$ make hello_world
cc $(pkg-config --cflags starpu-1.4) hello_world.c -o hello_world $(pkg-config --libs starpu-1.4)
$ ./hello_world
Hello world
Callback function (arg 42)
```

11.1.7 Where To Execute A Codelet

```
struct starpu_codelet cl =
{
    .where = STARPU_CPU,
    .cpu_funcs = { cpu_func },
    .nbuffers = 0
};
```

We create a codelet which may only be executed on the CPUs. The optional field starpu_codelet::where is a bitmask which defines where the codelet may be executed. Here, the value STARPU_CPU means that only CPUs can execute this codelet. When the optional field starpu_codelet::where is unset, its value is automatically set based on the availability of the different fields `XXX_funcs`.

11.2 Vector Scaling

The previous example has shown how to submit tasks. In this section, we show how StarPU tasks can manipulate data.
The full source code for this example is given in Full source code for the 'Scaling a Vector' example.

11.2.1 Source Code of Vector Scaling

Programmers can describe the data layout of their application so that StarPU is responsible for enforcing data coherency and availability across the machine. Instead of handling complex (and non-portable) mechanisms to perform data movements, programmers only declare which piece of data is accessed and/or modified by a task, and StarPU makes sure that when a computational kernel starts somewhere (e.g. on a GPU), its data are available locally.
Before submitting those tasks, programmers first need to declare the different pieces of data to StarPU using the functions `starpu_*_data_register`. To ease the development of applications for StarPU, it is possible to describe multiple types of data layout. A type of data layout is called an **interface**. There are different predefined interfaces available in StarPU, here we will consider the **vector interface**.
The following lines show how to declare an array of `NX` elements of type `float` using the vector interface:

```
float vector[NX];
starpu_data_handle_t vector_handle;
starpu_vector_data_register(&vector_handle, STARPU_MAIN_RAM, (uintptr_t)vector, NX, sizeof(vector[0]));
```

The first argument, called the **data handle**, is an opaque pointer which designates the array within StarPU. This is also the structure which is used to describe which data is used by a task. The second argument is the node number where the data originally resides. Here it is STARPU_MAIN_RAM since the array `vector` is in the main memory. Then comes the pointer `vector` where the data can be found in main memory, the number of elements in the vector and the size of each element. The following shows how to construct a StarPU task that will manipulate the vector and a constant factor.

```
float factor = 3.14;
struct starpu_task *task = starpu_task_create();
task->cl = &cl;                          /* Pointer to the codelet defined below */
task->handles[0] = vector_handle;        /* First parameter of the codelet */
task->cl_arg = &factor;
task->cl_arg_size = sizeof(factor);
task->synchronous = 1;
starpu_task_submit(task);
```

Since the factor is a mere constant float value parameter, it does not need a preliminary registration, and can just be passed through the pointer starpu_task::cl_arg like in the previous example. The vector parameter is described by its handle. starpu_task::handles should be set with the handles of the data, the access modes for the data are defined in the field starpu_codelet::modes (STARPU_R for read-only, STARPU_W for write-only and STARPU_RW for read and write access).

The definition of the codelet can be written as follows:

```
void scal_cpu_func(void *buffers[], void *cl_arg)
{
    unsigned i;
    float *factor = cl_arg;
    /* length of the vector */
    unsigned n = STARPU_VECTOR_GET_NX(buffers[0]);
    /* CPU copy of the vector pointer */
    float *val = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
    for (i = 0; i < n; i++)
        val[i] *= *factor;
}
struct starpu_codelet cl =
{
    .cpu_funcs = { scal_cpu_func },
    .nbuffers = 1,
    .modes = { STARPU_RW }
};
```

The first argument is an array that gives a description of all the buffers passed in the array starpu_task::handles. The size of this array is given by the field starpu_codelet::nbuffers. For the sake of genericity, this array contains pointers to the different interfaces describing each buffer. In the case of the **vector interface**, the location of the vector (resp. its length) is accessible in the starpu_vector_interface::ptr (resp. starpu_vector_interface::nx) of this interface. Since the vector is accessed in a read-write fashion, any modification will automatically affect future accesses to this vector made by other tasks.

The second argument of the function `scal_cpu_func` contains a pointer to the parameters of the codelet (given in starpu_task::cl_arg), so that we read the constant factor from this pointer.

11.2.2 Execution of Vector Scaling

```
$ make vector_scal
cc $(pkg-config --cflags starpu-1.4) vector_scal.c -o vector_scal $(pkg-config --libs starpu-1.4)
$ ./vector_scal
0.000000 3.000000 6.000000 9.000000 12.000000
```

11.3 Vector Scaling on an Hybrid CPU/GPU Machine

Contrary to the previous examples, the task submitted in this example may not only be executed by the CPUs, but also by a CUDA device.

11.3.1 Definition of the CUDA Kernel

The CUDA implementation can be written as follows. It needs to be compiled with a CUDA compiler such as nvcc, the NVIDIA CUDA compiler driver. It must be noted that the vector pointer returned by STARPU_VECTOR_GET_PTR is here a pointer in GPU memory, so that it can be passed as such to the kernel call `vector_mult_cuda`.

```
#include <starpu.h>
static __global__ void vector_mult_cuda(unsigned n, float *val, float factor)
{
        unsigned i =  blockIdx.x*blockDim.x + threadIdx.x;
        if (i < n)
               val[i] *= factor;
}
```

```
extern "C" void scal_cuda_func(void *buffers[], void *_args)
{
        float *factor = (float *)_args;
        /* length of the vector */
        unsigned n = STARPU_VECTOR_GET_NX(buffers[0]);
        /* local copy of the vector pointer */
        float *val = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
        unsigned threads_per_block = 64;
        unsigned nblocks = (n + threads_per_block-1) / threads_per_block;
        vector_mult_cuda«<nblocks,threads_per_block, 0, starpu_cuda_get_local_stream()»>(n, val, *factor);
        cudaError_t status = cudaGetLastError();
        if (status != cudaSuccess) STARPU_CUDA_REPORT_ERROR(status);
        cudaStreamSynchronize(starpu_cuda_get_local_stream());
}
```

11.3.2 Definition of the OpenCL Kernel

The OpenCL implementation can be written as follows. StarPU provides tools to compile a OpenCL kernel stored in a file.

```
__kernel void vector_mult_opencl(int nx, __global float* val, float factor)
{
        const int i = get_global_id(0);
        if (i < nx)
        {
                val[i] *= factor;
        }
}
```

Contrary to CUDA and CPU, STARPU_VECTOR_GET_DEV_HANDLE has to be used, which returns a `cl_mem` (which is not a device pointer, but an OpenCL handle), which can be passed as such to the OpenCL kernel. The difference is important when using partitioning, see Partitioning Data.

```
#include <starpu.h>
extern struct starpu_opencl_program programs;
void scal_opencl_func(void *buffers[], void *_args)
{
    float *factor = _args;
    int id, devid, err;                   /* OpenCL specific code */
    cl_kernel kernel;                     /* OpenCL specific code */
    cl_command_queue queue;               /* OpenCL specific code */
    cl_event event;                       /* OpenCL specific code */
    /* length of the vector */
    unsigned n = STARPU_VECTOR_GET_NX(buffers[0]);
    /* OpenCL copy of the vector pointer */
    cl_mem val = (cl_mem)STARPU_VECTOR_GET_DEV_HANDLE(buffers[0]);
    {  /* OpenCL specific code */
         id = starpu_worker_get_id();
         devid = starpu_worker_get_devid(id);
         err = starpu_opencl_load_kernel(&kernel, &queue, &programs,
                                         "vector_mult_opencl", /* Name of the codelet */
                                         devid);
         if (err != CL_SUCCESS) STARPU_OPENCL_REPORT_ERROR(err);
         err = clSetKernelArg(kernel, 0, sizeof(n), &n);
         err |= clSetKernelArg(kernel, 1, sizeof(val), &val);
         err |= clSetKernelArg(kernel, 2, sizeof(*factor), factor);
         if (err) STARPU_OPENCL_REPORT_ERROR(err);
    }
    {   /* OpenCL specific code */
        size_t global=n;
        size_t local;
        size_t s;
        cl_device_id device;
        starpu_opencl_get_device(devid, &device);
        err = clGetKernelWorkGroupInfo (kernel, device, CL_KERNEL_WORK_GROUP_SIZE, sizeof(local), &local,
      &s);
        if (err != CL_SUCCESS) STARPU_OPENCL_REPORT_ERROR(err);
        if (local > global) local=global;
        else global = (global + local-1) / local * local;
        err = clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &global, &local, 0, NULL, &event);
        if (err != CL_SUCCESS) STARPU_OPENCL_REPORT_ERROR(err);
    }
    {  /* OpenCL specific code */
         clFinish(queue);
         starpu_opencl_collect_stats(event);
         clReleaseEvent(event);
         starpu_opencl_release_kernel(kernel);
    }
}
```

11.3.3 Definition of the Main Code

The CPU implementation is the same as in the previous section.

Here is the source of the main application. You can notice that the fields starpu_codelet::cuda_funcs and starpu_codelet::opencl_funcs are set to define the pointers to the CUDA and OpenCL implementations of the task.

```
/*
* This example demonstrates how to use StarPU to scale an array by a factor.
* It shows how to manipulate data with StarPU's data management library.
*  1- how to declare a piece of data to StarPU (starpu_vector_data_register)
*  2- how to describe which data are accessed by a task (task->handles[0])
*  3- how a kernel can manipulate the data (buffers[0].vector.ptr)
*/
#include <starpu.h>
#define    NX    2048
extern void scal_cpu_func(void *buffers[], void *_args);
extern void scal_sse_func(void *buffers[], void *_args);
extern void scal_cuda_func(void *buffers[], void *_args);
extern void scal_opencl_func(void *buffers[], void *_args);
static struct starpu_codelet cl =
{
    .where = STARPU_CPU | STARPU_CUDA | STARPU_OPENCL,
    /* CPU implementation of the codelet */
    .cpu_funcs = { scal_cpu_func, scal_sse_func },
    .cpu_funcs_name = { "scal_cpu_func", "scal_sse_func" },
#ifdef STARPU_USE_CUDA
    /* CUDA implementation of the codelet */
    .cuda_funcs = { scal_cuda_func },
#endif
#ifdef STARPU_USE_OPENCL
    /* OpenCL implementation of the codelet */
    .opencl_funcs = { scal_opencl_func },
#endif
    .nbuffers = 1,
    .modes = { STARPU_RW }
};
#ifdef STARPU_USE_OPENCL
struct starpu_opencl_program programs;
#endif
int main(int argc, char **argv)
{
    /* We consider a vector of float that is initialized just as any of C
* data */
    float vector[NX];
    unsigned i;
    for (i = 0; i < NX; i++)
        vector[i] = 1.0f;
    fprintf(stderr, "BEFORE: First element was %f\n", vector[0]);
    /* Initialize StarPU with default configuration */
    starpu_init(NULL);
#ifdef STARPU_USE_OPENCL
    starpu_opencl_load_opencl_from_file("examples/basic_examples/vector_scal_opencl_kernel.cl", &programs,
      NULL);
#endif
    /* Tell StaPU to associate the "vector" vector with the "vector_handle"
* identifier.  When a task needs to access a piece of data, it should
* refer to the handle that is associated to it.
* In the case of the "vector" data interface:
*  - the first argument of the registration method is a pointer to the
*    handle that should describe the data
*  - the second argument is the memory node where the data (ie.  "vector")
*    resides initially:  STARPU_MAIN_RAM stands for an address in main memory, as
*    opposed to an address on a GPU for instance.
*  - the third argument is the address of the vector in RAM
*  - the fourth argument is the number of elements in the vector
*  - the fifth argument is the size of each element.
*/
    starpu_data_handle_t vector_handle;
    starpu_vector_data_register(&vector_handle, STARPU_MAIN_RAM, (uintptr_t)vector, NX, sizeof(vector[0]));
    float factor = 3.14;
    /* create a synchronous task:  any call to starpu_task_submit will block
* until it is terminated */
    struct starpu_task *task = starpu_task_create();
    task->synchronous = 1;
    task->cl = &cl;
    /* the codelet manipulates one buffer in RW mode */
    task->handles[0] = vector_handle;
    /* an argument is passed to the codelet, beware that this is a
* READ-ONLY buffer and that the codelet may be given a pointer to a
* COPY of the argument */
    task->cl_arg = &factor;
    task->cl_arg_size = sizeof(factor);
    /* execute the task on any eligible computational resource */
    starpu_task_submit(task);
    /* StarPU does not need to manipulate the array anymore so we can stop
* monitoring it */
    starpu_data_unregister(vector_handle);
#ifdef STARPU_USE_OPENCL
    starpu_opencl_unload_opencl(&programs);
#endif
    /* terminate StarPU, no task can be submitted after */
    starpu_shutdown();
    fprintf(stderr, "AFTER First element is %f\n", vector[0]);
    return 0;
}
```

11.3.4 Execution of Hybrid Vector Scaling

The Makefile given at the beginning of the section must be extended to give the rules to compile the CUDA source code. Note that the source file of the OpenCL kernel does not need to be compiled now, it will be compiled at runtime when calling the function starpu_opencl_load_opencl_from_file().

```
CFLAGS  += $(shell pkg-config --cflags starpu-1.4)
LDLIBS  += $(shell pkg-config --libs starpu-1.4)
CC       = gcc

vector_scal: vector_scal.o vector_scal_cpu.o vector_scal_cuda.o vector_scal_opencl.o

%.o: %.cu
       nvcc $(CFLAGS) $< -c $@

clean:
       rm -f vector_scal *.o
```

```
$ make
```

and to execute it, with the default configuration:

```
$ ./vector_scal
0.000000 3.000000 6.000000 9.000000 12.000000
```

or for example, by disabling CPU devices:

```
$ STARPU_NCPU=0 ./vector_scal
0.000000 3.000000 6.000000 9.000000 12.000000
```

or by disabling CUDA devices (which may permit to enable the use of OpenCL, see Enabling OpenCL) :

```
$ STARPU_NCUDA=0 ./vector_scal
0.000000 3.000000 6.000000 9.000000 12.000000
```

Chapter 12

Full Source Code for the 'Scaling a Vector' Example

12.1 Main Application

```
/*
* This example demonstrates how to use StarPU to scale an array by a factor.
* It shows how to manipulate data with StarPU's data management library.
*  1- how to declare a piece of data to StarPU (starpu_vector_data_register)
*  2- how to describe which data are accessed by a task (task->handles[0])
*  3- how a kernel can manipulate the data (buffers[0].vector.ptr)
*/
#include <starpu.h>
#define    NX    2048
extern void scal_cpu_func(void *buffers[], void *_args);
extern void scal_sse_func(void *buffers[], void *_args);
extern void scal_cuda_func(void *buffers[], void *_args);
extern void scal_opencl_func(void *buffers[], void *_args);
static struct starpu_codelet cl =
{
    .where = STARPU_CPU | STARPU_CUDA | STARPU_OPENCL,
    /* CPU implementation of the codelet */
    .cpu_funcs = { scal_cpu_func, scal_sse_func },
    .cpu_funcs_name = { "scal_cpu_func", "scal_sse_func" },
#ifdef STARPU_USE_CUDA
    /* CUDA implementation of the codelet */
    .cuda_funcs = { scal_cuda_func },
#endif
#ifdef STARPU_USE_OPENCL
    /* OpenCL implementation of the codelet */
    .opencl_funcs = { scal_opencl_func },
#endif
    .nbuffers = 1,
    .modes = { STARPU_RW }
};
#ifdef STARPU_USE_OPENCL
struct starpu_opencl_program programs;
#endif
int main(int argc, char **argv)
{
    /* We consider a vector of float that is initialized just as any of C
* data */
    float vector[NX];
    unsigned i;
    for (i = 0; i < NX; i++)
        vector[i] = 1.0f;
    fprintf(stderr, "BEFORE: First element was %f\n", vector[0]);
    /* Initialize StarPU with default configuration */
    starpu_init(NULL);
#ifdef STARPU_USE_OPENCL
    starpu_opencl_load_opencl_from_file("examples/basic_examples/vector_scal_opencl_kernel.cl", &programs,
      NULL);
#endif
    /* Tell StaPU to associate the "vector" vector with the "vector_handle"
* identifier.  When a task needs to access a piece of data, it should
* refer to the handle that is associated to it.
* In the case of the "vector" data interface:
*  - the first argument of the registration method is a pointer to the
*    handle that should describe the data
*  - the second argument is the memory node where the data (ie.  "vector")
*    resides initially:  STARPU_MAIN_RAM stands for an address in main memory, as
*    opposed to an address on a GPU for instance.
*  - the third argument is the address of the vector in RAM
*  - the fourth argument is the number of elements in the vector
*  - the fifth argument is the size of each element.
*/
    starpu_data_handle_t vector_handle;
    starpu_vector_data_register(&vector_handle, STARPU_MAIN_RAM, (uintptr_t)vector, NX, sizeof(vector[0]));
    float factor = 3.14;
    /* create a synchronous task:  any call to starpu_task_submit will block
* until it is terminated */
    struct starpu_task *task = starpu_task_create();
    task->synchronous = 1;
    task->cl = &cl;
    /* the codelet manipulates one buffer in RW mode */
    task->handles[0] = vector_handle;
    /* an argument is passed to the codelet, beware that this is a
* READ-ONLY buffer and that the codelet may be given a pointer to a
* COPY of the argument */
    task->cl_arg = &factor;
    task->cl_arg_size = sizeof(factor);
    /* execute the task on any eligible computational resource */
    starpu_task_submit(task);
    /* StarPU does not need to manipulate the array anymore so we can stop
* monitoring it */
    starpu_data_unregister(vector_handle);
#ifdef STARPU_USE_OPENCL
    starpu_opencl_unload_opencl(&programs);
#endif
    /* terminate StarPU, no task can be submitted after */
    starpu_shutdown();
    fprintf(stderr, "AFTER First element is %f\n", vector[0]);
    return 0;
}
```

12.2 CPU Kernel

```
#include <starpu.h>
#include <xmmintrin.h>
/* This kernel takes a buffer and scales it by a constant factor */
void scal_cpu_func(void *buffers[], void *cl_arg)
{
    unsigned i;
    float *factor = cl_arg;
    /*
* The "buffers" array matches the task->handles array:  for instance
* task->handles[0] is a handle that corresponds to a data with
* vector "interface", so that the first entry of the array in the
* codelet  is a pointer to a structure describing such a vector (ie.
* struct starpu_vector_interface *).  Here, we therefore manipulate
* the buffers[0] element as a vector:  nx gives the number of elements
* in the array, ptr gives the location of the array (that was possibly
* migrated/replicated), and elemsize gives the size of each elements.
*/
    struct starpu_vector_interface *vector = buffers[0];
    /* length of the vector */
    unsigned n = STARPU_VECTOR_GET_NX(vector);
    /* get a pointer to the local copy of the vector:  note that we have to
* cast it in (float *) since a vector could contain any type of
* elements so that the .ptr field is actually a uintptr_t */
    float *val = (float *)STARPU_VECTOR_GET_PTR(vector);
    /* scale the vector */
    for (i = 0; i < n; i++)
        val[i] *= *factor;
}
void scal_sse_func(void *buffers[], void *cl_arg)
{
    float *vector = (float *) STARPU_VECTOR_GET_PTR(buffers[0]);
    unsigned int n = STARPU_VECTOR_GET_NX(buffers[0]);
    unsigned int n_iterations = n/4;
    __m128 *VECTOR = (__m128*) vector;
    __m128 FACTOR STARPU_ATTRIBUTE_ALIGNED(16);
    float factor = *(float *) cl_arg;
    FACTOR = _mm_set1_ps(factor);
    unsigned int i;
    for (i = 0; i < n_iterations; i++)
        VECTOR[i] = _mm_mul_ps(FACTOR, VECTOR[i]);
    unsigned int remainder = n%4;
    if (remainder != 0)
    {
        unsigned int start = 4 * n_iterations;
        for (i = start; i < start+remainder; ++i)
        {
            vector[i] = factor * vector[i];
        }
    }
}
```

12.3 CUDA Kernel

```
#include <starpu.h>
static __global__ void vector_mult_cuda(unsigned n, float *val, float factor)
{
        unsigned i =  blockIdx.x*blockDim.x + threadIdx.x;
        if (i < n)
                val[i] *= factor;
}
extern "C" void scal_cuda_func(void *buffers[], void *_args)
{
        float *factor = (float *)_args;
        /* length of the vector */
        unsigned n = STARPU_VECTOR_GET_NX(buffers[0]);
        /* local copy of the vector pointer */
        float *val = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
        unsigned threads_per_block = 64;
        unsigned nblocks = (n + threads_per_block-1) / threads_per_block;
        vector_mult_cuda«<nblocks,threads_per_block, 0, starpu_cuda_get_local_stream()»>(n, val, *factor);
        cudaError_t status = cudaGetLastError();
        if (status != cudaSuccess) STARPU_CUDA_REPORT_ERROR(status);
        cudaStreamSynchronize(starpu_cuda_get_local_stream());
}
```

12.4 OpenCL Kernel

12.4.1 Invoking the Kernel

```
#include <starpu.h>
extern struct starpu_opencl_program programs;
void scal_opencl_func(void *buffers[], void *_args)
{
    float *factor = _args;
    int id, devid, err;                   /* OpenCL specific code */
    cl_kernel kernel;                     /* OpenCL specific code */
    cl_command_queue queue;               /* OpenCL specific code */
    cl_event event;                       /* OpenCL specific code */
    /* length of the vector */
    unsigned n = STARPU_VECTOR_GET_NX(buffers[0]);
    /* OpenCL copy of the vector pointer */
    cl_mem val = (cl_mem)STARPU_VECTOR_GET_DEV_HANDLE(buffers[0]);
    {  /* OpenCL specific code */
         id = starpu_worker_get_id();
         devid = starpu_worker_get_devid(id);
         err = starpu_opencl_load_kernel(&kernel, &queue, &programs,
                                         "vector_mult_opencl", /* Name of the codelet */
                                         devid);
         if (err != CL_SUCCESS) STARPU_OPENCL_REPORT_ERROR(err);
         err = clSetKernelArg(kernel, 0, sizeof(n), &n);
         err |= clSetKernelArg(kernel, 1, sizeof(val), &val);
         err |= clSetKernelArg(kernel, 2, sizeof(*factor), factor);
         if (err) STARPU_OPENCL_REPORT_ERROR(err);
    }
    {   /* OpenCL specific code */
        size_t global=n;
        size_t local;
        size_t s;
        cl_device_id device;
        starpu_opencl_get_device(devid, &device);
        err = clGetKernelWorkGroupInfo (kernel, device, CL_KERNEL_WORK_GROUP_SIZE, sizeof(local), &local,
      &s);
        if (err != CL_SUCCESS) STARPU_OPENCL_REPORT_ERROR(err);
        if (local > global) local=global;
        else global = (global + local-1) / local * local;
        err = clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &global, &local, 0, NULL, &event);
        if (err != CL_SUCCESS) STARPU_OPENCL_REPORT_ERROR(err);
    }
    {  /* OpenCL specific code */
         clFinish(queue);
         starpu_opencl_collect_stats(event);
         clReleaseEvent(event);
         starpu_opencl_release_kernel(kernel);
    }
}
```

12.4.2 Source of the Kernel

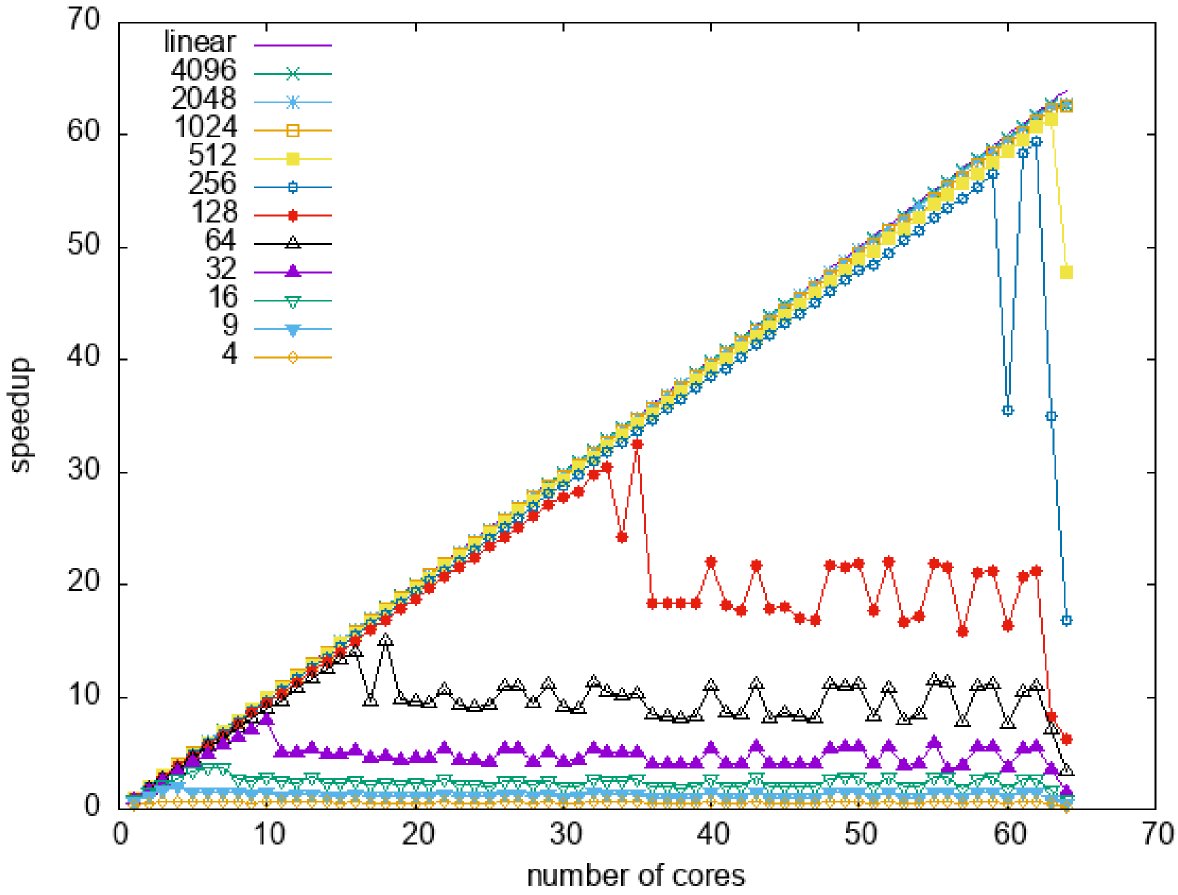
```
__kernel void vector_mult_opencl(int nx, __global float* val, float factor)
{
        const int i = get_global_id(0);
        if (i < nx)
        {
                val[i] *= factor;
        }
}
```

Chapter 13

Tasks In StarPU

13.1 Task Granularity

Similar to other runtimes, StarPU introduces some overhead in managing tasks. This overhead, while not always negligible, is mitigated by its intelligent scheduling and data management capabilities. The typical order of magnitude for this overhead is a few microseconds, which is notably smaller than the inherent CUDA overhead. To ensure that this overhead remains insignificant, the work assigned to a task should be substantial enough.
The length of tasks should ideally be relatively larger to effectively counterbalance this overhead. It iss advised to consider the offline performance feedback, which provides insights into task lengths. Monitoring task lengths becomes crucial if you're encountering suboptimal performance.
To gauge the scalability potential based task size, you can run the `tests/microbenchs/tasks_size_overhead.sh` script. It provides a visual representation of the speedup achievable with independent tasks of very small sizes.
This benchmark is installed in `$STARPU_PATH/lib/starpu/examples/`. It gives a glimpse into how long a task should be (in µs) for StarPU overhead to be low enough to keep efficiency. The script generates a plot illustrating the speedup trends for tasks of different sizes, correlated with the number of CPUs in use.
For example, in the figure below, for 128 µs tasks (the red line), StarPU overhead is low enough to guarantee a good speedup if the number of CPUs is not more than 36. But with the same number of CPUs, 64 µs tasks (the black line) cannot have a correct speedup. The number of CPUs must be decreased to about 17 in order to keep efficiency.



To determine the task size your application is using, it is possible to use `starpu_fxt_data_trace` as explained in Data trace and tasks length.

The selection of a scheduler in StarPU also plays a significant role. Different schedulers have varying impacts on the overall execution. For example, the `dmda` scheduler may require additional time to make decisions, while the `eager` scheduler tends to be more immediate in its decisions.

To assess the impact of scheduler choice on your target machine, you can once again utilize the `tasks_size_overhead.sh` script. This script provides valuable insights into how different schedulers affect performance in conjunction with task sizes.

13.2 Task Submission

To enable StarPU to perform online optimizations effectively, it is recommended to submit tasks asynchronously whenever possible. The goal is to maximize the level of asynchronous submission, allowing StarPU to have more flexibility in optimizing the scheduling process. Ideally, all tasks should be submitted asynchronously, and the use of functions like starpu_task_wait_for_all() or starpu_data_unregister() should be limited to waiting for task completion. StarPU will then be able to rework the whole schedule, overlap computation with communication, manage accelerator local memory usage, etc. A simple example is in the file `examples/basic_examples/variable.c`

13.3 Task Priorities

StarPU's default behavior considers tasks in the order they are submitted by the application. However, in scenarios where the application programmer possesses knowledge about certain tasks that should take priority due to their impact on performance (such as tasks whose output is crucial for subsequent tasks), the starpu_task::priority field can be utilized to convey this information to StarPU's scheduling process.

An example is provided in the application `examples/heat/dw_factolu_tag.c`.

13.4 Setting Many Data Handles For a Task

The maximum number of data that a task can manage is fixed by the macro STARPU_NMAXBUFS. This macro has a default value which can be customized through the `configure` option --enable-maxbuffers.
However, if you have specific cases where you need tasks to manage more data than the maximum allowed, you can use the field starpu_task::dyn_handles when defining a task, along with the field starpu_codelet::dyn_modes when defining the corresponding codelet.
This dynamic handle mechanism enables tasks to handle additional data beyond the usual limit imposed by STARPU_NMAXBUFS.

```
enum starpu_data_access_mode modes[STARPU_NMAXBUFS+1] =
{
        STARPU_R, STARPU_R, ...
};
struct starpu_codelet dummy_big_cl =
{
        .cuda_funcs = { dummy_big_kernel },
        .opencl_funcs = { dummy_big_kernel },
        .cpu_funcs = { dummy_big_kernel },
        .cpu_funcs_name = { "dummy_big_kernel" },
        .nbuffers = STARPU_NMAXBUFS+1,
        .dyn_modes = modes
};
task = starpu_task_create();
task->cl = &dummy_big_cl;
task->dyn_handles = malloc(task->cl->nbuffers * sizeof(starpu_data_handle_t));
for(i=0 ; i<task->cl->nbuffers ; i++)
{
        task->dyn_handles[i] = handle;
}
starpu_task_submit(task);
starpu_data_handle_t *handles = malloc(dummy_big_cl.nbuffers * sizeof(starpu_data_handle_t));
for(i=0 ; i<dummy_big_cl.nbuffers ; i++)
{
        handles[i] = handle;
}
starpu_task_insert(&dummy_big_cl,
                  STARPU_VALUE, &dummy_big_cl.nbuffers, sizeof(dummy_big_cl.nbuffers),
                  STARPU_DATA_ARRAY, handles, dummy_big_cl.nbuffers,
                  0);
```

The whole code for this complex data interface is available in the file `examples/basic_examples/dynamic_handles.c`.

13.5 Setting a Variable Number Of Data Handles For a Task

Normally, the number of data handles given to a task is set with starpu_codelet::nbuffers. This field can however be set to STARPU_VARIABLE_NBUFFERS, in which case starpu_task::nbuffers must be set, and starpu_task::modes (or starpu_task::dyn_modes, see Setting Many Data Handles For a Task) should be used to specify the modes for the handles. Examples in `examples/basic_examples/dynamic_handles.c` show how to implement it.

13.6 Insert Task Utility

StarPU provides the wrapper function starpu_task_insert() to ease the creation and submission of tasks.
Here is the implementation of a codelet:

```
void func_cpu(void *descr[], void *_args)
{
        int *x0 = (int *)STARPU_VARIABLE_GET_PTR(descr[0]);
        float *x1 = (float *)STARPU_VARIABLE_GET_PTR(descr[1]);
        int ifactor;
        float ffactor;
        starpu_codelet_unpack_args(_args, &ifactor, &ffactor);
        *x0 = *x0 * ifactor;
        *x1 = *x1 * ffactor;
}
struct starpu_codelet mycodelet =
{
        .cpu_funcs = { func_cpu },
        .cpu_funcs_name = { "func_cpu" },
        .nbuffers = 2,
        .modes = { STARPU_RW, STARPU_RW }
};
```

And the call to starpu_task_insert():

```
starpu_task_insert(&mycodelet,
                   STARPU_VALUE, &ifactor, sizeof(ifactor),
                   STARPU_VALUE, &ffactor, sizeof(ffactor),
                   STARPU_RW, data_handles[0],
                   STARPU_RW, data_handles[1],
                   0);
```

The call to starpu_task_insert() is equivalent to the following code:

```
struct starpu_task *task = starpu_task_create();
task->cl = &mycodelet;
task->handles[0] = data_handles[0];
task->handles[1] = data_handles[1];
char *arg_buffer;
size_t arg_buffer_size;
starpu_codelet_pack_args(&arg_buffer, &arg_buffer_size,
                    STARPU_VALUE, &ifactor, sizeof(ifactor),
                    STARPU_VALUE, &ffactor, sizeof(ffactor),
                    0);
task->cl_arg = arg_buffer;
task->cl_arg_size = arg_buffer_size;
int ret = starpu_task_submit(task);
```

In the example file `tests/main/insert_task_value.c`, we use these two ways to create and submit tasks. Instead of calling starpu_codelet_pack_args(), one can also call starpu_codelet_pack_arg_init(), then starpu_codelet_pack_arg() for each data, then starpu_codelet_pack_arg_fini() as follow:

```
struct starpu_task *task = starpu_task_create();
task->cl = &mycodelet;
task->handles[0] = data_handles[0];
task->handles[1] = data_handles[1];
struct starpu_codelet_pack_arg_data state;
starpu_codelet_pack_arg_init(&state);
starpu_codelet_pack_arg(&state, &ifactor, sizeof(ifactor));
starpu_codelet_pack_arg(&state, &ffactor, sizeof(ffactor));
starpu_codelet_pack_arg_fini(&state, &task->cl_arg, &task->cl_arg_size);
int ret = starpu_task_submit(task);
```

A full code example is in file `tests/main/pack.c`.

Here a similar call using STARPU_DATA_ARRAY.

```
starpu_task_insert(&mycodelet,
                   STARPU_DATA_ARRAY, data_handles, 2,
                   STARPU_VALUE, &ifactor, sizeof(ifactor),
                   STARPU_VALUE, &ffactor, sizeof(ffactor),
                   0);
```

If some part of the task insertion depends on the value of some computation, the macro STARPU_DATA_ACQUIRE_CB can be very convenient. For instance, assuming that the index variable `i` was registered as handle `A_handle[i]`:

```
/* Compute which portion we will work on, e.g. pivot */
starpu_task_insert(&which_index, STARPU_W, i_handle, 0);
/* And submit the corresponding task */
STARPU_DATA_ACQUIRE_CB(i_handle, STARPU_R,
                       starpu_task_insert(&work, STARPU_RW, A_handle[i], 0));
```

The macro STARPU_DATA_ACQUIRE_CB submits an asynchronous request for acquiring data `i` for the main application, and will execute the code given as the third parameter when it is acquired. In other words, as soon as the value of `i` computed by the codelet `which_index` can be read, the portion of code passed as the third parameter of STARPU_DATA_ACQUIRE_CB will be executed, and is allowed to read from `i` to use it e.g. as an index. Note that this macro is only available when compiling StarPU with the compiler `gcc`. In the example file `tests/datawizard/acquire_cb_insert.c`, this macro is used.

StarPU also provides a utility function starpu_codelet_unpack_args() to retrieve the STARPU_VALUE arguments passed to the task. There is several ways of calling starpu_codelet_unpack_args(). The full code examples are available in the file `tests/main/insert_task_value.c`.

```
void func_cpu(void *descr[], void *_args)
{
        int ifactor;
        float ffactor;
        starpu_codelet_unpack_args(_args, &ifactor, &ffactor);
}
void func_cpu(void *descr[], void *_args)
{
        int ifactor;
        float ffactor;
        starpu_codelet_unpack_args(_args, &ifactor, 0);
        starpu_codelet_unpack_args(_args, &ifactor, &ffactor);
}
void func_cpu(void *descr[], void *_args)
{
        int ifactor;
        float ffactor;
        char buffer[100];
        starpu_codelet_unpack_args_and_copyleft(_args, buffer, 100, &ifactor, 0);
        starpu_codelet_unpack_args(buffer, &ffactor);
}
```

Instead of calling starpu_codelet_unpack_args(), one can also call starpu_codelet_unpack_arg_init(), then starpu_codelet_pack_arg() or starpu_codelet_dup_arg() or starpu_codelet_pick_arg() for each data, then starpu_codelet_unpack_arg_fini() as follow:

```
void func_cpu(void *descr[], void *_args)
{
        int ifactor;
        float ffactor;
        size_t size = sizeof(int) + 2*sizeof(size_t) + sizeof(int) + sizeof(float);
        struct starpu_codelet_pack_arg_data state;
        starpu_codelet_unpack_arg_init(&state, _args, size);
        starpu_codelet_unpack_arg(&state, (void**)&ifactor, sizeof(ifactor));
        starpu_codelet_unpack_arg(&state, (void**)&ffactor, sizeof(ffactor));
        starpu_codelet_unpack_arg_fini(&state);
}
void func_cpu(void *descr[], void *_args)
{
        int *ifactor;
        float *ffactor;
        size_t size;
        size_t psize = sizeof(int) + 2*sizeof(size_t) + sizeof(int) + sizeof(float);
        struct starpu_codelet_pack_arg_data state;
        starpu_codelet_unpack_arg_init(&state, _args, psize);
        starpu_codelet_dup_arg(&state, (void**)&ifactor, &size);
        assert(size == sizeof(*ifactor));
        starpu_codelet_dup_arg(&state, (void**)&ffactor, &size);
        assert(size == sizeof(*ffactor));
        starpu_codelet_unpack_arg_fini(&state);
}
void func_cpu(void *descr[], void *_args)
{
        int *ifactor;
        float *ffactor;
        size_t size;
        size_t psize = sizeof(int) + 2*sizeof(size_t) + sizeof(int) + sizeof(float);
        struct starpu_codelet_pack_arg_data state;
        starpu_codelet_unpack_arg_init(&state, _args, psize);
        starpu_codelet_pick_arg(&state, (void**)&ifactor, &size);
        assert(size == sizeof(*ifactor));
        starpu_codelet_pick_arg(&state, (void**)&ffactor, &size);
        assert(size == sizeof(*ffactor));
        starpu_codelet_unpack_arg_fini(&state);
}
```

During unpacking one can also call starpu_codelet_unpack_discard_arg() to skip saving the argument in pointer. A full code example is in file `tests/main/pack.c`.

13.7 Other Task Utility Functions

Here a list of other functions to help with task management.

- The function starpu_task_dup() creates a duplicate of an existing task. The new task is identical to the original task in terms of its parameters, dependencies, and execution characteristics.
- The function starpu_task_set() is used to set the parameters of a task before it is executed, while starpu_task_build() is used to create a task with the specified parameters.

StarPU provides several functions to help insert data into a task. The function starpu_task_insert_data_make_room() is used to allocate memory space for a data structure that is required for inserting data into a task. This function is called before inserting any data handles into a task, and ensures that enough memory is available for the data to be stored. Once memory is allocated, the data handle can be inserted into the task using the following functions

- starpu_task_insert_data_process_arg() processes a scalar argument of a task and inserts it into the task's data structure. This function also performs any necessary data allocation and transfer operations.
- starpu_task_insert_data_process_array_arg() processes an array argument of a task and inserts it into the task's data structure. This function handles the allocation and transfer of the array data, as well as setting up the appropriate metadata to describe the array.
- starpu_task_insert_data_process_mode_array_arg() processes a mode array argument of a task and inserts it into the task's data structure. This function handles the allocation and transfer of the mode array data, as well as setting up the appropriate metadata to describe the mode array. Additionally, this function also computes the necessary sizes and strides for the data associated with the mode array argument.

Chapter 14

Data Management

TODO: intro which mentions consistency among other things

14.1 Data Interface

StarPU provides several data interfaces for programmers to describe the data layout of their application. There are predefined interfaces already available in StarPU. Users can define new data interfaces as explained in Defining A New Data Interface. All functions provided by StarPU are documented in Data Interfaces. You will find a short list below.

14.1.1 Variable Data Interface

A variable is a given-size byte element, typically a scalar. Here is an example of how to register a variable data to StarPU by using starpu_variable_data_register(). A full code example for the variable data interface is available in the file `examples/basic_examples/variable.c`.

```
float var = 42.0;
starpu_data_handle_t var_handle;
starpu_variable_data_register(&var_handle, STARPU_MAIN_RAM, (uintptr_t)&var, sizeof(var));
```

14.1.2 Vector Data Interface

A vector is a fixed number of elements of a given size. Here is an example of how to register a vector data to StarPU by using starpu_vector_data_register(). A full code example for the vector data interface is available in the file `examples/filters/fvector.c`.

```
float vector[NX];
starpu_data_handle_t vector_handle;
starpu_vector_data_register(&vector_handle, STARPU_MAIN_RAM, (uintptr_t)vector, NX, sizeof(vector[0]));
```

Vectors can be partitioned into pieces by using starpu_vector_filter_block(). They can also be partitioned with some overlapping by using starpu_vector_filter_block_shadow(). An example is in the file `examples/filters/shadow.c`.

By default, StarPU uses the same size for each piece. If different sizes are desired, starpu_vector_filter_list() or starpu_vector_filter_list_long() can be used instead.

To just divide in two pieces, starpu_vector_filter_divide_in_2() can be used.

In addition, contiguous variables can be picked from a vector by using starpu_vector_filter_pick_variable() with starpu_data_filter::get_child_ops set to starpu_vector_filter_pick_variable_child_ops(). An example is in the file `examples/filters/fvector_pick_variable.c`.

14.1.3 Matrix Data Interface

To register 2-D matrices with a potential padding, one can use the matrix data interface. Here is an example of how to register a matrix data to StarPU by using starpu_matrix_data_register().

A full code example for the matrix data interface is available in the file `examples/filters/fmatrix.c`.

```
float *matrix;
starpu_data_handle_t matrix_handle;
matrix = (float*)malloc(width * height * sizeof(float));
starpu_matrix_data_register(&matrix_handle, STARPU_MAIN_RAM, (uintptr_t)matrix, width, width, height,
      sizeof(float));
```

2D matrices can be partitioned into 2D matrices along the x dimension by using starpu_matrix_filter_block(), and along the y dimension by using starpu_matrix_filter_vertical_block().
They can also be partitioned with some overlapping by using starpu_matrix_filter_block_shadow() and starpu_matrix_filter_vertical_block_shadow(). An example is in the file `examples/filters/shadow2d.c`.
In addition, contiguous vectors can be picked from a matrix along the Y dimension by using starpu_matrix_filter_pick_vector_y() with starpu_data_filter::get_child_ops set to starpu_matrix_filter_pick_vector_child_ops(). An example is in the file `examples/filters/fmatrix_pick_vector.c`.
Variable can be also picked from a matrix by using starpu_matrix_filter_pick_variable() with starpu_data_filter::get_child_ops needs set to starpu_matrix_filter_pick_variable_child_ops(). An example is in the file `examples/filters/fmatrix_pick_variable.c`.

14.1.4 Block Data Interface

To register 3-D matrices with potential paddings on Y and Z dimensions, one can use the block data interface. Here is an example of how to register a block data to StarPU by using starpu_block_data_register(). A full code example for the block data interface is available in the file `examples/filters/fblock.c`.

```
float *block;
starpu_data_handle_t block_handle;
block = (float*)malloc(nx*ny*nz*sizeof(float));
starpu_block_data_register(&block_handle, STARPU_MAIN_RAM, (uintptr_t)block, nx, nx*ny, nx, ny, nz,
      sizeof(float));
```

3D matrices can be partitioned along the x dimension by using starpu_block_filter_block(), or along the y dimension by using starpu_block_filter_vertical_block(), or along the z dimension by using starpu_block_filter_depth_block().
They can also be partitioned with some overlapping by using starpu_block_filter_block_shadow(), starpu_block_filter_vertical_block_sh
or starpu_block_filter_depth_block_shadow(). An example is in the file `examples/filters/shadow3d.c`.
In addition, contiguous matrices can be picked from a block along the Z dimension or the Y dimension by using starpu_block_filter_pick_matrix_z() or starpu_block_filter_pick_matrix_y() with starpu_data_filter::get_child_ops set to starpu_block_filter_pick_matrix_child_ops(). An example is in the file `examples/filters/fblock_pick_matrix.c`.
Variable can be also picked from a block by using starpu_block_filter_pick_variable() with starpu_data_filter::get_child_ops set to starpu_block_filter_pick_variable_child_ops(). An example is in the file `examples/filters/fblock_pick_variable.c`.

14.1.5 Tensor Data Interface

To register 4-D matrices with potential paddings on Y, Z, and T dimensions, one can use the tensor data interface. Here is an example of how to register a tensor data to StarPU by using starpu_tensor_data_register(). A full code example for the tensor data interface is available in the file `examples/filters/ftensor.c`.

```
float *block;
starpu_data_handle_t block_handle;
block = (float*)malloc(nx*ny*nz*nt*sizeof(float));
starpu_tensor_data_register(&block_handle, STARPU_MAIN_RAM, (uintptr_t)block, nx, nx*ny, nx*ny*nz, nx, ny,
      nz, nt, sizeof(float));
```

4D matrices can be partitioned along the x dimension by using starpu_tensor_filter_block(), or along the y dimension by using starpu_tensor_filter_vertical_block(), or along the z dimension by using starpu_tensor_filter_depth_block(), or along the t dimension by using starpu_tensor_filter_time_block().
They can also be partitioned with some overlapping by using starpu_tensor_filter_block_shadow(), starpu_tensor_filter_vertical_block_
starpu_tensor_filter_depth_block_shadow(), or starpu_tensor_filter_time_block_shadow(). An example is in the file `examples/filters/shadow4d.c`.
In addition, contiguous blocks can be picked from a block along the T dimension, Z dimension or the Y dimension by using starpu_tensor_filter_pick_block_t(), starpu_tensor_filter_pick_block_z(), or starpu_tensor_filter_pick_block_y(), and starpu_data_filter::get_child_ops set to starpu_tensor_filter_pick_block_child_ops(). An example is in the file `examples/filters/ftensor_pick_block.c`.
Variable can be also picked from a tensor by using starpu_tensor_filter_pick_variable() with starpu_data_filter::get_child_ops set to starpu_tensor_filter_pick_variable_child_ops(). An example is in the file `examples/filters/ftensor_pick_variable.c`.

14.1.6 Ndim Data Interface

To register N-dim matrices, one can use the Ndim data interface. Here is an example of how to register a 5-dim data to StarPU by using starpu_ndim_data_register(). A full code example for the ndim data interface is available in the file `examples/filters/fndim.c`.

```
float *arr5d;
starpu_data_handle_t arr5d_handle;
starpu_malloc((void **)&arr5d, NX*NY*NZ*NT*NG*sizeof(float));
unsigned nn[5] = {NX, NY, NZ, NT, NG};
unsigned ldn[5] = {1, NX, NX*NY, NX*NY*NZ, NX*NY*NZ*NT};
starpu_ndim_data_register(&arr5d_handle, STARPU_MAIN_RAM, (uintptr_t)arr5d, ldn, nn, 5, sizeof(float));
```

N-dim matrices can be partitioned along the given dimension by using starpu_ndim_filter_block(). They can also be partitioned with some overlapping by using starpu_ndim_filter_block_shadow(). An example is in the file `examples/filters/shadownd.c`.

Taking into account existing data interfaces, there are several specialized functions which can partition a 0-dim array, 1-dim array, 2-dim array, 3-dim array or 4-dim array into

- variables by using starpu_ndim_filter_to_variable() and starpu_data_filter::get_child_ops set to starpu_ndim_filter_to_variable_c (see file `examples/filters/fndim_to_variable.c`),
- vectors by using starpu_ndim_filter_to_vector() and starpu_data_filter::get_child_ops set to starpu_ndim_filter_to_vector_child_ (see file `examples/filters/fndim_to_vector.c`),
- matrices by using starpu_ndim_filter_to_matrix() and starpu_data_filter::get_child_ops set to starpu_ndim_filter_to_matrix_child (see file `examples/filters/fndim_to_matrix.c`),
- blocks by using starpu_ndim_filter_to_block() and starpu_data_filter::get_child_ops set to starpu_ndim_filter_to_block_child_ops (see file `examples/filters/fndim_to_block.c`),
- or tensors by using starpu_ndim_filter_to_tensor() and starpu_data_filter::get_child_ops set to starpu_ndim_filter_to_tensor_chi (see file `examples/filters/fndim_to_tensor.c`).

In addition, contiguous (n-1)dim arrays can be picked from a ndim array along the given dimension by using starpu_ndim_filter_pick_ndim(). An example is in the file `examples/filters/fndim_pick_ndim.c`.

In specific cases which consider existing data interfaces, contiguous variables, vectors, matrices, blocks, or tensors can be along the given dimension picked from a

- 1-dim array by using starpu_ndim_filter_1d_pick_variable() and starpu_data_filter::get_child_ops set to starpu_ndim_filter_pick_variable_child_ops() (see file `examples/filters/fndim_1d_pick_variable.c`),
- 2-dim array by using starpu_ndim_filter_2d_pick_vector() and starpu_data_filter::get_child_ops set to starpu_ndim_filter_pick_vector_child_ops() (see file `examples/filters/fndim_2d_pick_vector.c`),
- 3-dim array by using starpu_ndim_filter_3d_pick_matrix() and starpu_data_filter::get_child_ops set to starpu_ndim_filter_pick_matrix_child_ops() (see file `examples/filters/fndim_3d_pick_matrix.c`),
- 4-dim array by using starpu_ndim_filter_4d_pick_block() and starpu_data_filter::get_child_ops set to starpu_ndim_filter_pick_block_child_ops() (see file `examples/filters/fndim_4d_pick_block.c`),
- or 5-dim array by using starpu_ndim_filter_5d_pick_tensor() and starpu_data_filter::get_child_ops set to starpu_ndim_filter_pick_tensor_child_ops() (see file `examples/filters/fndim_5d_pick_tensor.c`).

Variable can be also picked from a ndim array by using starpu_ndim_filter_pick_variable() with starpu_data_filter::get_child_ops set to starpu_ndim_filter_pick_variable_child_ops(). An example is in the file `examples/filters/fndim_pick_variable.c`.

14.1.7 BCSR Data Interface

BCSR (Blocked Compressed Sparse Row Representation) sparse matrix data can be registered to StarPU using the bcsr data interface. Here is an example on how to do so by using starpu_bcsr_data_register().

```
/*
* We use the following matrix:
*
*   +----------------+
*   |  0   1   0   0 |
*   |  2   3   0   0 |
*   |  4   5   8   9 |
*   |  6   7  10  11 |
*   +----------------+
*
* nzval  = [0, 1, 2, 3] ++ [4, 5, 6, 7] ++ [8, 9, 10, 11]
* colind = [0, 0, 1]
* rowptr = [0, 1, 3]
* r = c = 2
*/
/* Size of the blocks */
int R = 2;
int C = 2;
int NROWS = 2;
int NNZ_BLOCKS = 3;    /* out of 4 */
int NZVAL_SIZE = (R*C*NNZ_BLOCKS);
int nzval[NZVAL_SIZE]  =
{
        0, 1, 2, 3,    /* First block  */
        4, 5, 6, 7,    /* Second block */
        8, 9, 10, 11   /* Third block  */
};
uint32_t colind[NNZ_BLOCKS] =
{
        0, /* block-column index for first block in nzval */
        0, /* block-column index for second block in nzval */
        1  /* block-column index for third block in nzval */
};
uint32_t rowptr[NROWS+1] =
{
        0, / * block-index in nzval of the first block of the first row.  */
        1, / * block-index in nzval of the first block of the second row.  */
        NNZ_BLOCKS /* number of blocks, to allow an easier element's access for the kernels */
};
starpu_data_handle_t bcsr_handle;
starpu_bcsr_data_register(&bcsr_handle,
                          STARPU_MAIN_RAM,
                          NNZ_BLOCKS,
                          NROWS,
                          (uintptr_t) nzval,
                          colind,
                          rowptr,
                          0, /* firstentry */
                          R,
                          C,
                          sizeof(nzval[0]));
```

An example on how to deal with such matrices is in the file `examples/spmv/dw_block_spmv.c`.

BCSR data handles can be partitioned into its dense matrix blocks by using starpu_bcsr_filter_canonical_block(), or split into other BCSR data handles by using starpu_bcsr_filter_vertical_block() (but only split along the leading dimension is supported, i.e. along adjacent nnz blocks). starpu_data_filter::get_child_ops needs to be set to starpu_bcsr_filter_canonical_block_child_ops() and starpu_data_filter::get_nchildren set to starpu_bcsr_filter_canonical_block_get_nchildren(). An example is available in `tests/datawizard/bcsr.c`.

14.1.8 CSR Data Interface

TODO

To register a Compressed Sparse Row Representation (CSR) sparse matrix, one can use the CSR data interface. A full code example for the CSR data interface is available in the file `mpi/tests/datatypes.c` to show how to register a COO matrix data to StarPU by using starpu_csr_data_register().

CSR data handles can be partitioned into vertical CSR matrices by using starpu_csr_filter_vertical_block(). An example is available in the file `examples/spmv/spmv.c`.

14.1.9 COO Data Interface

To register 2-D matrices given in the coordinate format (COO), one can use the COO data interface. A full code example for the COO data interface is available in the file `tests/datawizard/interfaces/coo/coo_interface.c` to show how to register a COO matrix data to StarPU by using starpu_coo_data_register().

14.2 Partitioning Data

An existing piece of data can be partitioned in sub parts to be used by different tasks, for instance:

```
#define NX 1048576
#define PARTS 16
int vector[NX];
starpu_data_handle_t handle;
```

```
/* Declare data to StarPU */
starpu_vector_data_register(&handle, STARPU_MAIN_RAM, (uintptr_t)vector, NX, sizeof(vector[0]));
/* Partition the vector in PARTS sub-vectors */
struct starpu_data_filter f =
{
    .filter_func = starpu_vector_filter_block,
    .nchildren = PARTS
};
starpu_data_partition(handle, &f);
```

The handle of a sub-data block of a composite data block can be retrieved by calling starpu_data_get_child(). Or the task submission first retrieves the number of sub-data blocks in a composite data block by calling starpu_data_get_nb_children() and then uses the function starpu_data_get_sub_data() or starpu_data_vget_sub_data() to retrieve the sub-handles to be passed as tasks parameters.

```
/* Submit a task on each sub-vector */
for (i=0; i<starpu_data_get_nb_children(handle); i++)
{
    /* Get subdata number i (there is only 1 dimension) */
    starpu_data_handle_t sub_handle = starpu_data_get_sub_data(handle, 1, i);
    struct starpu_task *task = starpu_task_create();
    task->handles[0] = sub_handle;
    task->cl = &cl;
    task->synchronous = 1;
    task->cl_arg = &factor;
    task->cl_arg_size = sizeof(factor);
    starpu_task_submit(task);
}
```

Partitioning can be applied several times by using starpu_data_map_filters() or starpu_data_vmap_filters() or starpu_data_map_filters_parray() or starpu_data_map_filters_array(), see `examples/basic_examples/mult.c` and `examples/filters/`.

Wherever the whole piece of data is already available, the partitioning will be done in-place, i.e. without allocating new buffers but just using pointers inside the existing copy. This is particularly important to be aware of when using OpenCL, where the kernel parameters are not pointers, but `cl_mem` handles. The kernel thus needs to be also passed the offset within the OpenCL buffer:

```
void opencl_func(void *buffers[], void *cl_arg)
{
    cl_mem vector = (cl_mem) STARPU_VECTOR_GET_DEV_HANDLE(buffers[0]);
    unsigned offset = STARPU_BLOCK_GET_OFFSET(buffers[0]);
    ...
    clSetKernelArg(kernel, 0, sizeof(vector), &vector);
    clSetKernelArg(kernel, 1, sizeof(offset), &offset);
    ...
}
```

And the kernel has to shift from the pointer passed by the OpenCL driver:

```
__kernel void opencl_kernel(__global int *vector, unsigned offset)
{
    block = (__global void *)block + offset;
    ...
}
```

When the sub-data is not of the same type as the original data, the field starpu_data_filter::get_child_ops needs to be set appropriately for StarPU to know which type should be used.

starpu_data_unpartition() should be called in the end to collect back the sub-pieces of data into the original piece of data.

StarPU provides various interfaces and filters for matrices, vectors, etc., but applications can also write their own data interfaces and filters, see `examples/interface` and `examples/filters/custom_mf` for an example, and see Defining A New Data Interface and Defining A New Data Filter for documentation.

14.3 Asynchronous Partitioning

The partitioning functions described in the previous section are synchronous: starpu_data_partition() and starpu_data_unpartition() both wait for all the tasks currently working on the data. This can be a bottleneck for the application.

An asynchronous API also exists, it works only on handles with sequential consistency. The principle is to first plan the partitioning, which returns data handles of the partition, which are not functional yet. When submitting tasks, one can mix using the handles of the partition or the whole data. One can even partition recursively and mix using handles at different levels of the recursion. Of course, StarPU will have to introduce coherency synchronization.

`examples/filters/fmultiple_submit_implicit.c` is a complete example using this technique. One can also look at `examples/filters/fmultiple_submit_readonly.c` which contains the explicit coherency synchronization which are automatically introduced by StarPU for `examples/filters/fmultiple_submit_implicit.c`.

In short, we first register a matrix and plan the partitioning:

```
starpu_matrix_data_register(&handle, STARPU_MAIN_RAM, (uintptr_t)matrix, NX, NX, NY, sizeof(matrix[0]));
struct starpu_data_filter f_vert =
{
        .filter_func = starpu_matrix_filter_block,
        .nchildren = PARTS
};
starpu_data_partition_plan(handle, &f_vert, vert_handle);
```

starpu_data_partition_plan() returns the handles for the partition in `vert_handle`.

One can then submit tasks working on the main `handle`, and tasks working on the sub handles `vert_handle`. Between using the main handle and the handles `vert_handle`, StarPU will automatically call starpu_data_partition_submit() and starpu_data_unpartition_submit(). Or call starpu_data_partition_submit_sequential_consistency() and starpu_data_unpartition_submit_sequential_consistency() to specify the coherency to be used for the main handle, or call starpu_data_unpartition_submit_sequential_consistency_cb() to specify a call-back function for the unpartitiong task. One can also call starpu_data_partition_readonly_submit() and starpu_data_unpartition_readonly_submit() which do not guarantee coherency if the application attempts to write to the main handle or any of its sub-handles while a task is still running. However, in read-only case we can also call starpu_data_partition_readonly_submit_sequential_consistency() to specify the coherency to be used for the main handle, or call starpu_data_partition_readwrite_upgrade_submit() to upgrade the partitioning of a data handle from read-only to read-write mode for a specific sub-handle. If users want to specify that the data won't be touched in write mode anymore and use multiple partition of the data at the same time, they can call starpu_data_partition_readonly_downgrade_submit().

After the task has completed using the data partition, starpu_data_partition_clean() or starpu_data_partition_clean_node() is used to clean up a data partition on the local node or on a specific node.

All this code is asynchronous, just submitting which tasks, partitioning and unpartitioning will be done at runtime.

Planning several partitioning of the same data is also possible, StarPU will unpartition and repartition as needed when mixing accesses of different partitions. If data access is done in read-only mode, StarPU will allow the different partitioning to coexist. As soon as a data is accessed in read-write mode, StarPU will automatically unpartition everything and activate only the partitioning leading to the data being written to.

For instance, for a stencil application, one can split a subdomain into its interior and halos, and then just submit a task updating the whole subdomain, then submit MPI sends/receives to update the halos, then submit again a task updating the whole subdomain, etc. and StarPU will automatically partition/unpartition each time.

14.4 Commute Data Access

By default, the implicit dependencies computed from data access use the sequential semantic. Notably, write accesses are always serialized in the order of submission. In some applicative cases, the write contributions can actually be performed in any order without affecting the eventual result. In this case, it is useful to drop the strictly sequential semantic, to improve parallelism by allowing StarPU to reorder the write accesses. This can be done by using the data access flag STARPU_COMMUTE. Accesses without this flag will however properly be serialized against accesses with this flag. For instance:

```
starpu_task_insert(&cl1, STARPU_R, h, STARPU_RW, handle, 0);
starpu_task_insert(&cl2, STARPU_R, handle1, STARPU_RW|STARPU_COMMUTE, handle, 0);
starpu_task_insert(&cl2, STARPU_R, handle2, STARPU_RW|STARPU_COMMUTE, handle, 0);
starpu_task_insert(&cl3, STARPU_R, g, STARPU_RW, handle, 0);
```

The two tasks running `cl2` will be able to commute: depending on whether the value of `handle1` or `handle2` becomes available first, the corresponding task running `cl2` will start first. The task running `cl1` will however always be run before them, and the task running `cl3` will always be run after them.

`tests/datawizard/commute2.c` is a complete example using the data access flag.

If a lot of tasks use the commute access on the same set of data and a lot of them are ready at the same time, it may become interesting to use an arbiter, see Concurrent Data Accesses.

14.5 Data Reduction

In various cases, some piece of data is used to accumulate intermediate results. For instances, the dot product of a vector, maximum/minimum finding, the histogram of a picture, etc. When these results are produced along the whole machine, it would not be efficient to accumulate them in only one place, incurring data transmission each and access concurrency.

StarPU provides a mode STARPU_REDUX, which permits to optimize this case: it will allocate a buffer on each worker (lazily), and accumulate intermediate results there. When the data is eventually accessed in the normal mode STARPU_R, StarPU will collect the intermediate results in just one buffer.

The function starpu_data_set_reduction_methods() must be called to specify how to initialize these buffers, and how to assemble partial results. The function starpu_data_set_reduction_methods_with_args() can also be used to pass arguments to the reduction and init tasks.

For instance, `examples/cg/cg.c` uses that to optimize its dot product: it first defines the codelets for initialization and reduction:

```
struct starpu_codelet bzero_variable_cl =
{
        .cpu_funcs = { bzero_variable_cpu },
        .cpu_funcs_name = { "bzero_variable_cpu" },
        .cuda_funcs = { bzero_variable_cuda },
        .nbuffers = 1,
}
static void accumulate_variable_cpu(void *descr[], void *cl_arg)
{
        double *v_dst = (double *)STARPU_VARIABLE_GET_PTR(descr[0]);
        double *v_src = (double *)STARPU_VARIABLE_GET_PTR(descr[1]);
        *v_dst = *v_dst + *v_src;
}
static void accumulate_variable_cuda(void *descr[], void *cl_arg)
{
        double *v_dst = (double *)STARPU_VARIABLE_GET_PTR(descr[0]);
        double *v_src = (double *)STARPU_VARIABLE_GET_PTR(descr[1]);
        cublasaxpy(1, (double)1.0, v_src, 1, v_dst, 1);
        cudaStreamSynchronize(starpu_cuda_get_local_stream());
}
struct starpu_codelet accumulate_variable_cl =
{
        .cpu_funcs = { accumulate_variable_cpu },
        .cpu_funcs_name = { "accumulate_variable_cpu" },
        .cuda_funcs = { accumulate_variable_cuda },
        .nbuffers = 2,
        .modes = {STARPU_RW|STARPU_COMMUTE, STARPU_R},
}
```

and attaches them as reduction methods for its handle `dtq`:

```
starpu_variable_data_register(&dtq_handle, -1, NULL, sizeof(type));
starpu_data_set_reduction_methods(dtq_handle, &accumulate_variable_cl, &bzero_variable_cl);
```

and `dtq_handle` can now be used with the mode STARPU_REDUX for the dot products with partitioned vectors:

```
for (b = 0; b < nblocks; b++)
    starpu_task_insert(&dot_kernel_cl,
        STARPU_REDUX, dtq_handle,
        STARPU_R, starpu_data_get_sub_data(v1, 1, b),
        STARPU_R, starpu_data_get_sub_data(v2, 1, b),
        0);
```

During registration, we have here provided `NULL`, i.e. there is no initial value to be taken into account during reduction. StarPU will thus only take into account the contributions from the tasks `dot_kernel_cl`. Also, it will not allocate any memory for `dtq_handle` before the tasks `dot_kernel_cl` are ready to run.

If another dot product has to be performed, one could unregister `dtq_handle`, and re-register it. But one can also call starpu_data_deinitialize_submit() or even starpu_data_invalidate_submit() with the parameter `dtq_handle`, which will clear all data from the handle, thus resetting it back to the initial status `register(NULL)`.

The example `examples/cg/cg.c` also uses reduction for the blocked gemv kernel, leading to yet more relaxed dependencies and more parallelism.

STARPU_REDUX can also be passed to starpu_mpi_task_insert() in the MPI case. This will however not produce any MPI communication, but just pass STARPU_REDUX to the underlying starpu_task_insert(). starpu_mpi_redux_data() posts tasks which will reduce the partial results among MPI nodes into the MPI node which owns the data. The function can be called by users to benefit from fine-tuning such as priority setting. If users do not call this function, StarPU wraps up reduction patterns automatically. The following example shows a hypothetical application which collects partial results into data `res`, then uses it for other computation, before looping again with a new reduction where the wrap-up of the reduction pattern is explicit:

```
for (i = 0; i < 100; i++)
{
    starpu_mpi_task_insert(MPI_COMM_WORLD, &init_res, STARPU_W, res, 0);
    starpu_mpi_task_insert(MPI_COMM_WORLD, &work, STARPU_RW, A, STARPU_R, B, STARPU_REDUX, res, 0);
    starpu_mpi_redux_data(MPI_COMM_WORLD, res);
    starpu_mpi_task_insert(MPI_COMM_WORLD, &work2, STARPU_RW, B, STARPU_R, res, 0);
}
```

starpu_mpi_redux_data() is called automatically in various cases, including when a task reading the reduced handle is inserted through starpu_mpi_task_insert(). The previous example could avoid calling starpu_mpi_redux_data(). Default priority (0) is used. The reduction tree arity is decided based on the size of the data to reduce: a flat tree is used with a small data (default to less than 1024 bytes), a binary tree otherwise. If the environment variable STARPU_MPI_REDUX_ARITY_THRESHOLD is set, the threshold between the size of a small data and a bigger data is modified. If the value is set to be negative, flat trees will always be used. If the value is set to 0, binary trees are used. Otherwise, the size of the data is compared to the size in the environment variable. Remaining distributed-memory reduction patterns are wrapped-up at the end of an application when calling starpu_mpi_wait_for_all().

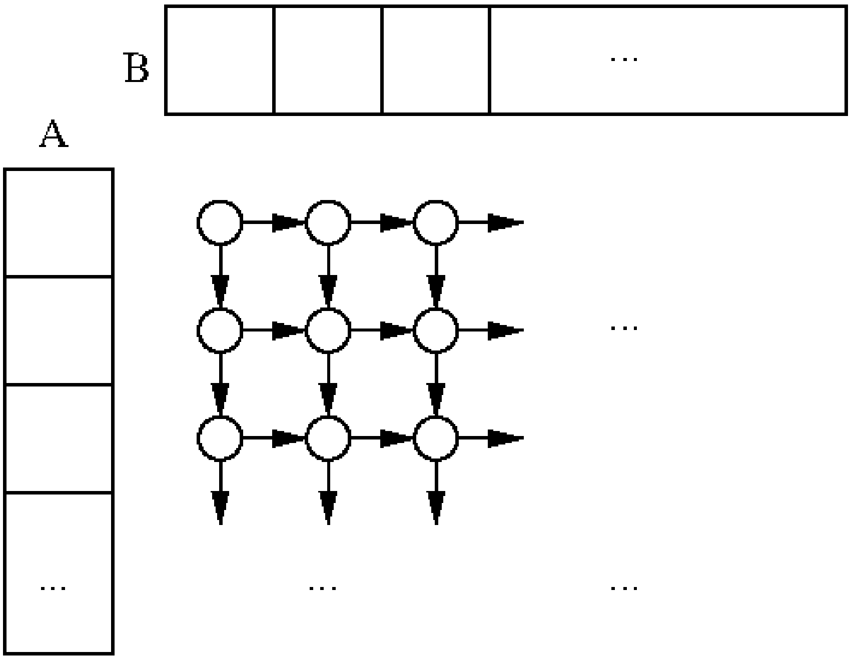
More details about MPI reduction are show in Section Inter-node reduction, and some examples for MPI data reduction are available in `mpi/examples/mpi_redux/`.

14.6 Concurrent Data Accesses

When several tasks are ready and will work on several data, StarPU is faced with the classical Dining Philosopher's problem, and has to determine the order in which it will run the tasks.
Data accesses usually use sequential ordering, so data accesses are usually already serialized, and thus by default, StarPU uses the Dijkstra solution which scales very well in terms of overhead: tasks will just acquire data one by one by data handle pointer value order.
When sequential ordering is disabled or the flag STARPU_COMMUTE is used, there may be a lot of concurrent accesses to the same data, and the Dijkstra solution gets only poor parallelism, typically in some pathological cases which do happen in various applications, for instance

```
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
        task[i][j] = starpu_task_build(&cl, STARPU_RW|STARPU_COMMUTE, A[i], STARPU_RW|STARPU_COMMUTE, B[j],
      0);
```

It creates a series of tasks that are completely parallel in terms of tasks dependencies thanks to commutation, but StarPU still has to prevent two tasks from operating on the same data. The Dijkstra solution here leads to a worst-case: the `task`[0][j] tasks will wait for each other since they all access the same `A`[0]. And `task`[1][0] will wait for `task`[0][0] because they both access the same `B`[0], `task`[1][1] will wait for `task`[0][1] because of `B`[1], etc. In the end, no parallism is achieved:



In this case, one can use a data access arbiter starpu_arbiter_t, which implements the classical centralized solution for the Dining Philosophers problem. One can call starpu_arbiter_create() to create a data access arbiter, and starpu_data_assign_arbiter() to make access to handle managed by arbiter. Once the application no longer needs the arbiter, one can call starpu_arbiter_destroy() to destroy the arbiter after all data assigned to the arbiter have been unregistered. This is more expensive in terms of overhead since it is centralized, but it opportunistically gets a lot of parallelism. The centralization can also be avoided by using several arbiters, thus separating sets of data for which arbitration will be done. If a task accesses data from different arbiters, it will acquire them arbiter by arbiter, in arbiter pointer value order.
See the `tests/datawizard/test_arbiter.cpp` example.
Arbiters however do not support the flag STARPU_REDUX yet.

14.7 Temporary Buffers

There are two kinds of temporary buffers: temporary data which just pass results from a task to another, and scratch data which are needed only internally by tasks.

14.7.1 Temporary Data

Data can be produced by a task, and consumed by another task, without being used by other parts of the application. In such case, registration can be done without prior allocation, by using the special memory node number `-1`, and passing a NULL pointer. StarPU will actually allocate memory only when the task creating the content gets scheduled, and destroy it on unregistration.

As the application will not use the data, it can be tedious for the application to have to unregister it. The unregistration can be done lazily by using the function starpu_data_unregister_submit(), which will record that no other tasks accessing the handle will be submitted, so that it can be freed as soon as the last task accessing it is completed.

The following code examplifies both points: it registers the temporary data, submits three tasks accessing it, and records the data for automatic unregistration.

```
starpu_vector_data_register(&handle, -1, NULL, n, sizeof(float));
starpu_task_insert(&produce_data, STARPU_W, handle, 0);
starpu_task_insert(&compute_data, STARPU_RW, handle, 0);
starpu_task_insert(&summarize_data, STARPU_R, handle, STARPU_W, result_handle, 0);
starpu_data_unregister_submit(handle);
```

The application may also want for the temporary data to be initialized on the fly before being used by the task. This can be done by using starpu_data_set_reduction_methods() to set an initialization codelet (no redux codelet is needed).

14.7.2 Scratch Data

Some kernels sometimes need temporary data to complete the computations, like a workspace. The application could allocate it at the start of the codelet function, and free it at the end, but this would be costly. It could also allocate one buffer per worker (similarly to How To Initialize A Computation Library Once For Each Worker?), but this would make them systematic and permanent. A more optimized way is to use the data access mode STARPU_SCRATCH, as examplified below, which provides per-worker buffers without content consistency. The buffer is registered only once, using memory node `-1`, i.e. the application didn't allocate memory for it, and StarPU will allocate it on demand at task execution.

```
starpu_variable_data_register(&workspace, -1, NULL, sizeof(float));
for (i = 0; i < N; i++)
    starpu_task_insert(&compute, STARPU_R, input[i], STARPU_SCRATCH, workspace, STARPU_W, output[i], 0);
```

StarPU will make sure that the buffer is allocated before executing the task, and make this allocation per-worker: for CPU workers, notably, each worker has its own buffer. This means that each task submitted above will actually have its own workspace, which will actually be the same for all tasks running one after the other on the same worker. Also, if for instance memory becomes scarce, StarPU will notice that it can free such buffers easily, since the content does not matter.

The example `examples/pi` uses scratches for some temporary buffer.

It may be useful to additionally use the STARPU_NOFOOTPRINT flag, when this buffer may have various size depending e.g. on specific CUDA versions or devices, to make it simpler to use performance models for simulated execution. See for instance `examples/cholesky/cholesky_kernels.c`

Chapter 15

Scheduling

15.1 Task Scheduling Policies

The basics of the scheduling policy are the following:

- The scheduler gets to schedule tasks (`push` operation) when they become ready to be executed, i.e. they are not waiting for some tags, data dependencies or task dependencies.
- Workers pull tasks (`pop` operation) one by one from the scheduler.

This means scheduling policies usually contain at least one queue of tasks to store them between the time when they become available, and the time when a worker gets to grab them.
By default, StarPU uses the work-stealing scheduler **lws**. This is because it provides correct load balance and locality even if the application codelets do not have performance models. Other non-modelling scheduling policies can be selected among the list below, thanks to the environment variable STARPU_SCHED. For instance, `export STARPU_SCHED=dmda` . Use `help` to get the list of available schedulers.
The function starpu_sched_get_predefined_policies() returns a NULL-terminated array of all predefined scheduling policies that are available in StarPU. Functions starpu_sched_get_sched_policy_in_ctx() and starpu_sched_get_sched_policy() return the scheduling policy of a task within a specific context or a default context, respectively.

15.1.1 Non Performance Modelling Policies

- The **eager** scheduler uses a central task queue, from which all workers draw tasks to work on concurrently. This however does not permit to prefetch data since the scheduling decision is taken late. If a task has a non-0 priority, it is put at the front of the queue.
- The **random** scheduler uses a queue per worker, and distributes tasks randomly according to assumed worker overall performance.
- The **ws** (work stealing) scheduler uses a queue per worker, and schedules a task on the worker which released it by default. When a worker becomes idle, it steals a task from the most loaded worker.
- The **lws** (locality work stealing) scheduler uses a queue per worker, and schedules a task on the worker which released it by default. When a worker becomes idle, it steals a task from neighbor workers. It also takes priorities into account.
- The **prio** scheduler also uses a central task queue, but sorts tasks by priority specified by the application.
- The **heteroprio** scheduler uses different priorities for the different processing units. This scheduler must be configured to work correctly and to expect high-performance as described in the corresponding section.

15.1.2 Performance Model-Based Task Scheduling Policies

If (**and only if**) your **codelets have performance models** (Performance Model Example), you should change the scheduler thanks to the environment variable STARPU_SCHED, to select one of the policies below, in order to take

advantage of StarPU's performance modelling. For instance, `export STARPU_SCHED=dmda` . Use `help` to get the list of available schedulers.
Note: Depending on the performance model type chosen, some preliminary calibration runs may be needed for the model to converge. If the calibration has not been done, or is insufficient yet, or if no performance model is specified for a codelet, every task built from this codelet will be scheduled using an **eager** fallback policy.
Troubleshooting: Configuring and recompiling StarPU using the `configure` option --enable-verbose displays some statistics at the end of execution about the percentage of tasks which have been scheduled by a DM$*$ family policy using performance model hints. A low or zero percentage may be the sign that performance models are not converging or that codelets do not have performance models enabled.

- The **dm** (deque model) scheduler takes task execution performance models into account to perform a HEFT-similar scheduling strategy: it schedules tasks where their termination time will be minimal. The difference with HEFT is that **dm** schedules tasks as soon as they become available, and thus in the order they become available, without taking priorities into account.
- The **dmda** (deque model data aware) scheduler is similar to **dm**, but it also takes data transfer time into account.
- The **dmdap** (deque model data aware prio) scheduler is similar to **dmda**, except that it sorts tasks by priority order, which allows becoming even closer to HEFT by respecting priorities after having made the scheduling decision (but it still schedules tasks in the order they become available).
- The **dmdar** (deque model data aware ready) scheduler is similar to **dmda**, but it also privileges tasks whose data buffers are already available on the target device.
- The **dmdas** combines **dmdap** and **dmdar:** it sorts tasks by priority order, but for a given priority it will privilege tasks whose data buffers are already available on the target device.
- The **dmdasd** (deque model data aware sorted decision) scheduler is similar to dmdas, except that when scheduling a task, it takes into account its priority when computing the minimum completion time, since this task may get executed before others, and thus the latter should be ignored.
- The **heft** (heterogeneous earliest finish time) scheduler is a deprecated alias for **dmda**.
- The **pheft** (parallel HEFT) scheduler is similar to **dmda**, it also supports parallel tasks (still experimental). It should not be used when several contexts using it are being executed simultaneously.
- The **peager** (parallel eager) scheduler is similar to eager, it also supports parallel tasks (still experimental). It should not be used when several contexts using it are being executed simultaneously.

15.1.3 Modularized Schedulers

StarPU provides a powerful way to implement schedulers, as documented in Defining A New Modular Scheduling Policy. It is currently shipped with the following pre-defined Modularized Schedulers :

- **modular-eager** , **modular-eager-prefetching** are eager-based Schedulers (without and with prefetching), they are naive schedulers, which try to map a task on the first available resource they find. The prefetching variant queues several tasks in advance to be able to do data prefetching. This may however degrade load balancing a bit.
- **modular-prio**, **modular-prio-prefetching**, **modular-eager-prio** are prio-based Schedulers (without / with prefetching):, similar to Eager-Based Schedulers. They can handle tasks which have a defined priority and schedule them accordingly. The **modular-eager-prio** variant integrates the eager and priority queue in a single component. This allows it to do a better job at pushing tasks.
- **modular-random**, **modular-random-prio**, **modular-random-prefetching**, **modular-random-prio-prefetching** are random-based Schedulers (without/with prefetching) : Select randomly a resource to be mapped on for each task.
- **modular-ws**) implements Work Stealing: Maps tasks to workers in round-robin, but allows workers to steal work from other workers.

- **modular-heft**, **modular-heft2**, and **modular-heft-prio** are HEFT Schedulers :
  Maps tasks to workers using a heuristic very close to Heterogeneous Earliest Finish Time. It needs that every task submitted to StarPU have a defined performance model (Performance Model Calibration) to work efficiently, but can handle tasks without a performance model. **modular-heft** just takes tasks by order. **modular-heft2** takes at most 5 tasks of the same priority and checks which one fits best. **modular-heft-prio** is similar to **modular-heft**, but only decides the memory node, not the exact worker, just pushing tasks to one central queue per memory node. By default, they sort tasks by priorities and privilege, running first a task which has most of its data already available on the target. These can however be changed with STARPU_SCHED_SORTED_ABOVE, STARPU_SCHED_SORTED_BELOW, and STARPU_SCHED_READY .
- **modular-heteroprio** is a Heteroprio Scheduler:
  Maps tasks to worker similarly to HEFT, but first attribute accelerated tasks to GPUs, then not-so-accelerated tasks to CPUs.

15.2 Task Distribution Vs Data Transfer

Distributing tasks to balance the load induces data transfer penalty. StarPU thus needs to find a balance between both. The target function that the scheduler **dmda** of StarPU tries to minimize is `alpha * T_execution + beta * T_data_transfer`, where `T_execution` is the estimated execution time of the codelet (usually accurate), and `T_data_transfer` is the estimated data transfer time. The latter is estimated based on bus calibration before execution start, i.e. with an idle machine, thus without contention. You can force bus re-calibration by running the tool `starpu_calibrate_bus`. The beta parameter defaults to `1`, but it can be worth trying to tweak it by using `export STARPU_SCHED_BETA=2` (STARPU_SCHED_BETA) for instance, since during real application execution, contention makes transfer times bigger. This is of course imprecise, but in practice, a rough estimation already gives the good results that a precise estimation would give.

Chapter 16

Examples in StarPU Sources

We have already seen some examples in Chapter Basic Examples. A tutorial is also installed in the directory `share/doc/starpu/tutorial/`.
Many examples are also available in the StarPU sources in the directory `examples/`. Simple examples include:

`incrementer/` Trivial incrementation test.

`basic_examples/` Simple documented Hello world and vector/scalar product (as shown in Basic Examples), matrix product examples (as shown in Performance Model Example), an example using the blocked matrix data interface, an example using the variable data interface, and an example using different formats on CPUs and GPUs.

`matvecmult/` OpenCL example from NVidia, adapted to StarPU.

`axpy/` AXPY CUBLAS operation adapted to StarPU.

`native_fortran/` Example of using StarPU's native Fortran support.

`fortran90/` Example of Fortran 90 bindings, using C marshalling wrappers.

`fortran/` Example of Fortran 77 bindings, using C marshalling wrappers.

More advanced examples include:

`filters/` Examples using filters, as shown in Partitioning Data.

`lu/` LU matrix factorization, see for instance `xlu_implicit.c`

`cholesky/` Cholesky matrix factorization, see for instance `cholesky_implicit.c`.

Part III

StarPU Applications

Chapter 17

Organization

This part presents how to write a StarPU application from an existing application.

Some of the applications presented in the following chapters and some others are available in the git repository `https://gitlab.inria.fr/starpu/starpu-applications`

Chapter 18

A Vector Scaling Application

18.1 Base version

The non-StarPU version shows a basic example that we will be using to illustrate how to use StarPU. It simply allocates a vector, and calls a scaling function over it.

```
void vector_scal_cpu(float *val, unsigned n, float factor)
{
        unsigned i;
        for (i = 0; i < n; i++)
                val[i] *= factor;
}
#define    NX    2048
int main(void)
{
        float *vector;
        unsigned i;
        vector = malloc(sizeof(vector[0]) * NX);
        for (i = 0; i < NX; i++)
                vector[i] = 1.0f;
        fprintf(stderr, "BEFORE : First element was %f\n", vector[0]);
        float factor = 3.14;
        vector_scal_cpu(vector, NX, factor);
        fprintf(stderr, "AFTER First element is %f\n", vector[0]);
        free(vector);
        return 0;
}
```

18.2 StarPU C version

18.2.1 Computation Kernels

We are going to transform here the computation function `vector_scal_cpu`.

```
void vector_scal_cpu(float *val, unsigned n, float factor)
{
        unsigned i;
        for (i = 0; i < n; i++)
                val[i] *= factor;
}
```

The StarPU corresponding function takes as parameters a list of DSM interfaces and a non-DSM parameter.

```
void vector_scal_cpu(void *buffers[], void *cl_arg)
{
```

The first DSM parameter is the vector and is available through `buffer`[0]. StarPU provides functions to get the vector data, and extract the pointer and size of the vector.

```
        struct starpu_vector_interface *vector = buffers[0];
        float *val = (float *)STARPU_VECTOR_GET_PTR(vector);
        unsigned n = STARPU_VECTOR_GET_NX(vector);
```

The non-DSM parameters are stored in the second argument of the function, and need to be unpacked.

```
        float factor;
        starpu_codelet_unpack_args(cl_arg, &factor);
```

It is then possible to perform the vector scaling as in the original function.

```
        unsigned i;
        for (i = 0; i < n; i++)
                val[i] *= factor;
```

Original code

```
void vector_scal_cpu(float *val, unsigned n, float
      factor)
{
 unsigned i;
 for (i = 0; i < n; i++)
 val[i] *= factor;
}
```

StarPU code

```
void vector_scal_cpu(void *buffers[], void *cl_arg)
{
 struct starpu_vector_interface *vector =
      buffers[0];
 float *val = (float
      *)STARPU_VECTOR_GET_PTR(vector);
 unsigned n = STARPU_VECTOR_GET_NX(vector);
 float factor;
 starpu_codelet_unpack_args(cl_arg, &factor);
 unsigned i;
 for (i = 0; i < n; i++)
 val[i] *= factor;
}
```

The GPU and OpenCL implementations can be seen in Full source code for the ’Scaling a Vector’ example.

18.2.2 Main Code

Let's look now at the main code.

- The `cl` codelet structure simply gathers pointers on the functions mentioned above, and notes that the functions takes only one DSM parameter.

  ```
  static struct starpu_codelet cl =
  {
          .cpu_funcs = {vector_scal_cpu},
          .cuda_funcs = {vector_scal_cuda},
          .opencl_funcs = {vector_scal_opencl},
          .nbuffers = 1,
          .modes = {STARPU_RW}
  };
  ```

- The `main` function starts with initializing StarPU with the default parameters.

  ```
          int ret = starpu_init(NULL);
          STARPU_CHECK_RETURN_VALUE(ret, "starpu_init");
  ```

- It then allocates the vector and fills it like the original code.

  ```
          vector = malloc(sizeof(vector[0]) * NX);
          for (i = 0; i < NX; i++)
                  vector[i] = 1.0f;
          fprintf(stderr, "BEFORE : First element was %f\n", vector[0]);
  ```

- It then registers the data to StarPU, and gets back a DSM handle. From now on, the application is not supposed to access `vector` directly, since its content may be copied and modified by a task on a GPU, the main-memory copy then being outdated.

  ```
          starpu_data_handle_t vector_handle;
          starpu_vector_data_register(&vector_handle, STARPU_MAIN_RAM, (uintptr_t)vector, NX,
  sizeof(vector[0]));
  ```

- It then submits a (asynchronous) task to StarPU.

  ```
          float factor = 3.14;
          ret = starpu_task_insert(&cl,
                                   STARPU_RW, vector_handle,
                                   STARPU_VALUE, &factor, sizeof(factor),
                                   0);
          STARPU_CHECK_RETURN_VALUE(ret, "starpu_task_insert");
  ```

- It waits for task completion, and unregisters the vector from StarPU, which brings back the modified version to main memory, so the result can be read.

  ```
          starpu_task_wait_for_all();
          starpu_data_unregister(vector_handle);
  ```

- Eventually, it shuts down StarPU:

  ```
          starpu_shutdown();
  ```

Original code

```
#define NX 2048
int main(void)
{
 float *vector;
 unsigned i;
 vector = malloc(sizeof(vector[0]) * NX);
 for (i = 0; i < NX; i++)
 vector[i] = 1.0f;
 fprintf(stderr, "BEFORE : First element was %f\n",
      vector[0]);
 float factor = 3.14;
 vector_scal_cpu(vector, NX, factor);
 fprintf(stderr, "AFTER First element is %f\n",
      vector[0]);
 free(vector);
 return 0;
}
```

StarPU code

```
#include <starpu.h>
extern void vector_scal_cpu(void *buffers[], void
      *_args);
extern void vector_scal_cuda(void *buffers[], void
      *_args);
extern void vector_scal_opencl(void *buffers[],
      void *_args);
static struct starpu_codelet cl =
{
 .cpu_funcs = {vector_scal_cpu},
 .cuda_funcs = {vector_scal_cuda},
 .opencl_funcs = {vector_scal_opencl},
 .nbuffers = 1,
 .modes = {STARPU_RW}
};
#ifdef STARPU_USE_OPENCL
struct starpu_opencl_program programs;
#endif
#define NX 2048
int main(void)
{
 float *vector;
 unsigned i;
 int ret = starpu_init(NULL);
 STARPU_CHECK_RETURN_VALUE(ret, "starpu_init");
#ifdef STARPU_USE_OPENCL
 starpu_opencl_load_opencl_from_file("vector_scal_opencl_kernel.cl",
      &programs, NULL);
#endif
 vector = malloc(sizeof(vector[0]) * NX);
 for (i = 0; i < NX; i++)
 vector[i] = 1.0f;
 fprintf(stderr, "BEFORE : First element was %f\n",
      vector[0]);
 starpu_data_handle_t vector_handle;
 starpu_vector_data_register(&vector_handle,
      STARPU_MAIN_RAM, (uintptr_t)vector, NX,
      sizeof(vector[0]));
 float factor = 3.14;
 ret = starpu_task_insert(&cl,
 STARPU_RW, vector_handle,
 STARPU_VALUE, &factor, sizeof(factor),
 0);
 STARPU_CHECK_RETURN_VALUE(ret,
      "starpu_task_insert");
 starpu_task_wait_for_all();
 starpu_data_unregister(vector_handle);
 fprintf(stderr, "AFTER First element is %f\n",
      vector[0]);
 free(vector);
#ifdef STARPU_USE_OPENCL
 starpu_opencl_unload_opencl(&programs);
#endif
 starpu_shutdown();
 return 0;
}
```

18.3 Building and Running

We will use the StarPU docker image.

```
$ docker run -it registry.gitlab.inria.fr/starpu/starpu-docker/starpu:latest
```

If your machine has GPU devices, you can use the following command to enable the GPU devices within the docker image.

```
$ docker run -it --gpus all registry.gitlab.inria.fr/starpu/starpu-docker/starpu:latest
```

From your docker image, you can then call the following commands.

```
$ cd src/starpu/doc/tutorial
$ make vector_scal
$ ./vector_scal
```

You can set the environment variable STARPU_WORKER_STATS to `1` when running your application to see the number of tasks executed by each device.

```
$ STARPU_WORKER_STATS=1 ./vector_scal
```

If your machine has GPU devices, you can force the execution on the GPU devices by setting the number of CPU workers to 0.

```
# to force the implementation on a GPU device, by default, it will enable CUDA
$ STARPU_WORKER_STATS=1 STARPU_NCPU=0 ./vector_scal

# to force the implementation on a OpenCL device
$ STARPU_WORKER_STATS=1 STARPU_NCPU=0 STARPU_NCUDA=0 ./vector_scal
```

Chapter 19

A Stencil Application

19.1 The Original Application

```
#define _(row,col,ld) ((row)+(col)*(ld))
void stencil5_cpu(double *xy, double *xm1y, double *xp1y, double *xym1, double *xyp1)
{
  *xy = (*xy + *xm1y + *xp1y + *xym1 + *xyp1) / 5;
}
int main(int argc, char **argv)
{
  int niter, n;
  int x, y, loop;
  read_params(argc, argv, &n, &niter);
  double *A = calloc(n*n, sizeof(*A));
  fill(A, n, n);
  for(loop=0 ; loop<niter; loop++)
  {
    for (x = 0; x < n; x++)
    {
      for (y = 0; y < n; y++)
      {
        int xm1 = (x==0) ?  n-1 :  x-1;
        int xp1 = (x==n-1) ?  0 :  x+1;
        int ym1 = (y==0) ?  n-1 :  y-1;
        int yp1 = (y==n-1) ?  0 :  y+1;
        stencil5_cpu(&A[_(x,y,n)],
                     &A[_(xm1,y,n)], &A[_(xp1,y,n)],
                     &A[_(x,ym1,n)], &A[_(x,yp1,n)]);
      }
    }
  }
  return 0;
}
```

19.2 The StarPU Application

The computation function must be defined through a codelet.

```
#define _(row,col,ld) ((row)+(col)*(ld))
void stencil5_cpu(void *descr[], void *_args)
{
  (void)_args;
  double *xy = (double *)STARPU_VARIABLE_GET_PTR(descr[0]);
  double *xm1y = (double *)STARPU_VARIABLE_GET_PTR(descr[1]);
  double *xp1y = (double *)STARPU_VARIABLE_GET_PTR(descr[2]);
  double *xym1 = (double *)STARPU_VARIABLE_GET_PTR(descr[3]);
  double *xyp1 = (double *)STARPU_VARIABLE_GET_PTR(descr[4]);
  *xy = (*xy + *xm1y + *xp1y + *xym1 + *xyp1) / 5;
}
struct starpu_codelet stencil5_cl =
{
  .cpu_funcs = {stencil5_cpu},
  .nbuffers = 5,
  .modes = {STARPU_RW, STARPU_R, STARPU_R, STARPU_R, STARPU_R},
  .model = &starpu_perfmodel_nop,
};
```

Data must be registered to StarPU.

```
  data_handles = malloc(n*n*sizeof(*data_handles));
  for(x = 0; x < n; x++)
  {
    for (y = 0; y < n; y++)
    {
      starpu_variable_data_register(&data_handles[_(x,y,n)],
                                    STARPU_MAIN_RAM,
                                    (uintptr_t)&(A[_(x,y,n)]), sizeof(double));
    }
  }
```

Instead of directly calling the function, a StarPU task must be created.

```
        int xm1 = (x==0) ?  n-1 :  x-1;
        int xp1 = (x==n-1) ?  0 :  x+1;
        int ym1 = (y==0) ?  n-1 :  y-1;
        int yp1 = (y==n-1) ?  0 :  y+1;
        starpu_task_insert(&stencil5_cl,
                           STARPU_RW, data_handles[_(x,y,n)],
                           STARPU_R, data_handles[_(xm1,y,n)],
                           STARPU_R, data_handles[_(xp1,y,n)],
                           STARPU_R, data_handles[_(x,ym1,n)],
                           STARPU_R, data_handles[_(x,yp1,n)],
                           0);
```

And finally data must be released from StarPU.

```
  for(x = 0; x < n; x++)
  {
    for (y = 0; y < n; y++)
    {
      starpu_data_unregister(data_handles[_(x,y,n)]);
    }
  }
```

The whole StarPU application looks as follows.

```
#define _(row,col,ld) ((row)+(col)*(ld))
void stencil5_cpu(void *descr[], void *_args)
{
  (void)_args;
  double *xy = (double *)STARPU_VARIABLE_GET_PTR(descr[0]);
  double *xm1y = (double *)STARPU_VARIABLE_GET_PTR(descr[1]);
  double *xp1y = (double *)STARPU_VARIABLE_GET_PTR(descr[2]);
  double *xym1 = (double *)STARPU_VARIABLE_GET_PTR(descr[3]);
  double *xyp1 = (double *)STARPU_VARIABLE_GET_PTR(descr[4]);
  *xy = (*xy + *xm1y + *xp1y + *xym1 + *xyp1) / 5;
}
struct starpu_codelet stencil5_cl =
{
  .cpu_funcs = {stencil5_cpu},
  .nbuffers = 5,
  .modes = {STARPU_RW, STARPU_R, STARPU_R, STARPU_R, STARPU_R},
  .model = &starpu_perfmodel_nop,
};
int main(int argc, char **argv)
{
  starpu_data_handle_t *data_handles;
  int ret;
  int niter, n;
  int x, y, loop;
  ret = starpu_init(NULL);
  STARPU_CHECK_RETURN_VALUE(ret, "starpu_init");
  read_params(argc, argv, &verbose, &n, &niter);
  double *A = calloc(n*n, sizeof(*A));
  fill(A, n, n);
  data_handles = malloc(n*n*sizeof(*data_handles));
  for(x = 0; x < n; x++)
  {
    for (y = 0; y < n; y++)
    {
      starpu_variable_data_register(&data_handles[_(x,y,n)],
                                    STARPU_MAIN_RAM,
                                    (uintptr_t)&(A[_(x,y,n)]), sizeof(double));
    }
  }
  for(loop=0 ; loop<niter; loop++)
  {
    for (x = 0; x < n; x++)
    {
      for (y = 0; y < n; y++)
      {
        int xm1 = (x==0) ?  n-1 :  x-1;
        int xp1 = (x==n-1) ?  0 :  x+1;
        int ym1 = (y==0) ?  n-1 :  y-1;
        int yp1 = (y==n-1) ?  0 :  y+1;
        starpu_task_insert(&stencil5_cl,
                           STARPU_RW, data_handles[_(x,y,n)],
                           STARPU_R, data_handles[_(xm1,y,n)],
                           STARPU_R, data_handles[_(xp1,y,n)],
                           STARPU_R, data_handles[_(x,ym1,n)],
                           STARPU_R, data_handles[_(x,yp1,n)],
                           0);
      }
    }
  }
  starpu_task_wait_for_all();
  for(x = 0; x < n; x++)
  {
    for (y = 0; y < n; y++)
    {
      starpu_data_unregister(data_handles[_(x,y,n)]);
    }
  }
  starpu_shutdown();
  return 0;
}
```

19.3 The StarPU MPI Application

The initialisation for StarPU-MPI is as follows.

```
  int ret = starpu_mpi_init_conf(&argc, &argv, 1, MPI_COMM_WORLD, NULL);
  STARPU_CHECK_RETURN_VALUE(ret, "starpu_mpi_init_conf");
  starpu_mpi_comm_rank(MPI_COMM_WORLD, &my_rank);
  starpu_mpi_comm_size(MPI_COMM_WORLD, &size);
```

An additional call to starpu_mpi_data_register() is necessary.

```
      starpu_variable_data_register(&data_handles[_(x,y,n)],
                                    STARPU_MAIN_RAM,
                                    (uintptr_t)&(A[_(x,y,n)]), sizeof(double));
      int mpi_rank = my_distrib(x, y, size);
      starpu_mpi_data_register(data_handles[_(x,y,n)], (y*n)+x, mpi_rank);
```

And to insert a task, the function starpu_mpi_task_insert() must be used.

```
        starpu_mpi_task_insert(MPI_COMM_WORLD, &stencil5_cl,
                               STARPU_RW, data_handles[_(x,y,n)],
                               STARPU_R, data_handles[_(xm1,y,n)],
                               STARPU_R, data_handles[_(xp1,y,n)],
                               STARPU_R, data_handles[_(x,ym1,n)],
                               STARPU_R, data_handles[_(x,yp1,n)],
                               0);
```

The whole StarPU-MPI application looks as follows.

```
#define _(row,col,ld) ((row)+(col)*(ld))
void stencil5_cpu(void *descr[], void *_args); // Same as in sequential StarPU
struct starpu_codelet stencil5_cl;  // Same as in sequential StarPU
/* Returns the MPI node number where data indexes index is */
int my_distrib(int x, int y, int nb_nodes)
{
  return ((int)(x / sqrt(nb_nodes) + (y / sqrt(nb_nodes)) * sqrt(nb_nodes))) % nb_nodes;
}
int main(int argc, char **argv)
{
  starpu_data_handle_t *data_handles;
  int niter, n;
  int my_rank, size, x, y, loop;
  int ret = starpu_mpi_init_conf(&argc, &argv, 1, MPI_COMM_WORLD, NULL);
  STARPU_CHECK_RETURN_VALUE(ret, "starpu_mpi_init_conf");
  starpu_mpi_comm_rank(MPI_COMM_WORLD, &my_rank);
  starpu_mpi_comm_size(MPI_COMM_WORLD, &size);
  read_params(argc, argv, &n, &niter);
  double *A = calloc(n*n, sizeof(*A));
  fill(A, n, n);
  data_handles = malloc(n*n*sizeof(*data_handles));
  for(x = 0; x < n; x++)
  {
    for (y = 0; y < n; y++)
    {
      starpu_variable_data_register(&data_handles[_(x,y,n)],
                                    STARPU_MAIN_RAM,
                                    (uintptr_t)&(A[_(x,y,n)]), sizeof(double));
      int mpi_rank = my_distrib(x, y, size);
      starpu_mpi_data_register(data_handles[_(x,y,n)], (y*n)+x, mpi_rank);
    }
  }
  for(loop=0 ; loop<niter; loop++)
  {
    for (x = 0; x < n; x++)
    {
      for (y = 0; y < n; y++)
      {
        int xm1 = (x==0) ?  n-1 :  x-1;
        int xp1 = (x==n-1) ?  0 :  x+1;
        int ym1 = (y==0) ?  n-1 :  y-1;
        int yp1 = (y==n-1) ?  0 :  y+1;
        starpu_mpi_task_insert(MPI_COMM_WORLD, &stencil5_cl,
                               STARPU_RW, data_handles[_(x,y,n)],
                               STARPU_R, data_handles[_(xm1,y,n)],
                               STARPU_R, data_handles[_(xp1,y,n)],
                               STARPU_R, data_handles[_(x,ym1,n)],
                               STARPU_R, data_handles[_(x,yp1,n)],
                               0);
      }
    }
  }
  starpu_task_wait_for_all();
  /* bring data back to node 0 and unregister it */
  for(x = 0; x < n; x++)
  {
    for (y = 0; y < n; y++)
    {
        starpu_mpi_data_migrate(MPI_COMM_WORLD, data_handles[_(x,y,n)], 0);
        starpu_data_unregister(data_handles[_(x,y,n)]);
    }
  }
  starpu_mpi_shutdown();
  return 0;
}
```

19.4 Running the application

```
$ docker run -it registry.gitlab.inria.fr/starpu/starpu-docker/starpu:latest
```

If your machine has GPU devices, you can use the following command to enable the GPU devices within the docker image.

```
$ docker run -it --gpus all registry.gitlab.inria.fr/starpu/starpu-docker/starpu:latest
```

From your docker image, you can then call the following commands.

```
$ git clone https://gitlab.inria.fr/starpu/starpu-applications.git
$ cd starpu-applications/stencil5
$ make
```

To run the non-StarPU application

```
$ ./stencil5 -v
```

To run the sequential StarPU application

```
$ ./stencil5_starpu -v
```

To run the StarPU MPI application. Setting the variable STARPU_COMM_STATS to 1 will display the amount of communication between the different MPI processes.

```
$ STARPU_COMM_STATS=1 mpirun -np 4 ./stencil5_starpu_mpi -v 4 3
```

Part IV

StarPU Performances

Chapter 20

Organization

This part shows how to measure application performances.

- Chapter Benchmarking StarPU introduces some interesting benchmarks which can be found in StarPU sources.
- Chapter Online Performance Tools gives information on online performance monitoring tools to help you analyze your program
- Chapter Offline Performance Tools gives information on offline performance tools such as a FxT library to trace execution data and tasks and a StarPU Eclipse Plugin to visualize data traces directly from the Eclipse IDE.

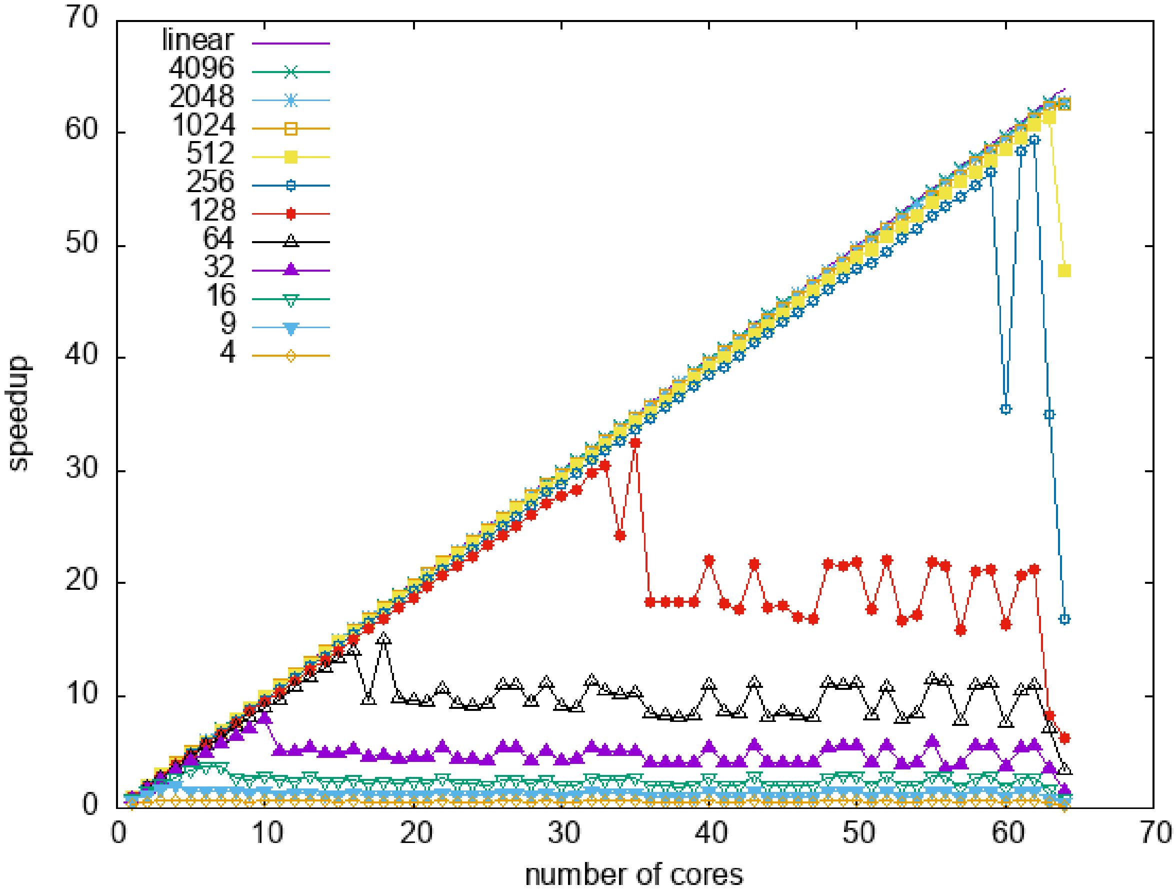
Chapter 21

Benchmarking StarPU

Some interesting benchmarks are installed among examples in `$STARPU_PATH/lib/starpu/examples/`. Make sure to try various schedulers, for instance `STARPU_SCHED=dmda`.

21.1 Task Size Overhead

This benchmark gives a glimpse into how long a task should be (in µs) for StarPU overhead to be low enough to keep efficiency. Running `tasks_size_overhead.sh` generates a plot of the speedup of tasks of various sizes, depending on the number of CPUs being used.



21.2 Data Transfer Latency

`local_pingpong` performs a ping-pong between the first two CUDA nodes, and prints the measured latency.

21.3 Matrix-Matrix Multiplication

`sgemm` and `dgemm` perform a blocked matrix-matrix multiplication using BLAS and cuBLAS. They output the obtained GFlops.

21.4 Cholesky Factorization

`cholesky_*` perform a Cholesky factorization (single precision). They use different dependency primitives.

21.5 LU Factorization

`lu_*` perform an LU factorization. They use different dependency primitives.

21.6 Simulated Benchmarks

It can also be convenient to try simulated benchmarks, if you want to give a try at CPU-GPU scheduling without actually having a GPU at hand. This can be done by using the SimGrid version of StarPU: first install the SimGrid simulator from `https://simgrid.org/` (we tested with SimGrid from 3.11 to 3.16, and 3.18 to 3.30. SimGrid versions 3.25 and above need to be configured with `-Denable_msg=ON`. Other versions may have compatibility issues, 3.17 notably does not build at all. MPI simulation does not work with version 3.22). Then configure StarPU with --enable-simgrid and rebuild and install it, and then you can simulate the performance for a few virtualized systems shipped along StarPU: attila, mirage, idgraf, and sirocco.
For instance:

```
$ export STARPU_PERF_MODEL_DIR=$STARPU_PATH/share/starpu/perfmodels/sampling
$ export STARPU_HOSTNAME=attila
$ $STARPU_PATH/lib/starpu/examples/cholesky_implicit -size $((960*20)) -nblocks 20
```

Will show the performance of the cholesky factorization with the attila system. It will be interesting to try with different matrix sizes and schedulers.
Performance models are available for `cholesky_*`, `lu_*`, `*gemm`, with block sizes 320, 640, or 960 (plus 1440 for sirocco), and for `stencil` with block size 128x128x128, 192x192x192, and 256x256x256.
Read Chapter SimGrid Support for more information on the SimGrid support.

Chapter 22

Online Performance Tools

22.1 On-line Performance Feedback

Some examples which apply online performance monitoring are in the directory `tests/perfmodels/`

22.1.1 Enabling On-line Performance Monitoring

In order to enable online performance monitoring, the application can call starpu_profiling_status_set() with the parameter STARPU_PROFILING_ENABLE. It is possible to detect whether monitoring is already enabled or not by calling starpu_profiling_status_get(). Enabling monitoring also reinitialize all previously collected feedback. The environment variable STARPU_PROFILING can also be set to `1` to achieve the same effect. The function starpu_profiling_init() can also be called during the execution to reinitialize performance counters and to start the profiling if the environment variable STARPU_PROFILING is set to `1`.
Likewise, performance monitoring is stopped by calling starpu_profiling_status_set() with the parameter STARPU_PROFILING_DISABLE. Note that this does not reset the performance counters so that the application may consult them later on.
More details about the performance monitoring API are available in Profiling.

22.1.2 Per-task Feedback

If profiling is enabled, a pointer to a structure starpu_profiling_task_info is put in the field starpu_task::profiling_info when a task terminates. This structure is automatically destroyed when the task structure is destroyed, either automatically or by calling starpu_task_destroy().
The structure starpu_profiling_task_info indicates the date when the task was submitted (starpu_profiling_task_info::submit_time), started (starpu_profiling_task_info::start_time), and terminated (starpu_profiling_task_info::end_time), relative to the initialization of StarPU with starpu_init(). User can call starpu_timing_timespec_delay_us() to calculate the time elapsed between start time and end time in microseconds. It also specifies the identifier of the worker that has executed the task (starpu_profiling_task_info::workerid). These dates are stored as `timespec` structures which users may convert into micro-seconds using the helper function starpu_timing_timespec_to_us(). User can call starpu_worker_get_current_task_exp_end() to get the date when the current task is expected to be finished.
When ::STARPU_ENERGY_PROFILING is enabled, starpu_profiling_task_info::energy_consumed, provides the amount of Joules used by the task.
It is worth noting that the application may directly access this structure from the callback executed at the end of the task. The structure starpu_task associated to the callback currently being executed is indeed accessible with the function starpu_task_get_current().

22.1.3 Per-codelet Feedback

The field starpu_codelet::per_worker_stats is an array of counters. Unless the STARPU_CODELET_PROFILING environment variable was set to 0, the `i`-th entry of the array is incremented every time a task implementing the codelet is executed on the `i`-th worker. This array is not reinitialized when profiling is enabled or disabled. The function starpu_codelet_display_stats() can be used to display the execution statistics of a specific codelet.

22.1.4 Per-worker Feedback

The second argument returned by the function starpu_profiling_worker_get_info() is a structure starpu_profiling_worker_info that gives statistics about the specified worker. This structure specifies:

- In starpu_profiling_worker_info::start_time, when StarPU started collecting profiling information for that worker.
- In starpu_profiling_worker_info::total_time, the duration of the profiling measurement interval.
- In starpu_profiling_worker_info::executed_tasks, the number of tasks that were executed while profiling was enabled.

It also specifies how much time was spent in various states (executing a task, executing a callback, waiting for a data transfer to complete, etc.). Since these can happen at the same time (waiting for a data transfer while executing the previous tasks, and scheduling the next task), we provide two views. Firstly, the "all" view:

- In starpu_profiling_worker_info::all_executing_time, the time spent executing kernels, thus real useful work.
- In starpu_profiling_worker_info::all_callback_time, the time spent executing application callbacks.
- In starpu_profiling_worker_info::all_waiting_time, the time spent waiting for data transfers.
- In starpu_profiling_worker_info::all_sleeping_time, the time spent during which there was no task to be executed, i.e. lack of parallelism.
- In starpu_profiling_worker_info::all_scheduling_time, the time spent scheduling tasks.

But these times overlap, notably with GPUs the schedulers runs while tasks are getting executed. Another view is the "split" view, which eliminates the overlapping, by considering for instance that it does not matter what is happening while tasks are getting executed, that should be accounted for "executing" time, and e.g. only the scheduling periods that happen while no task is getting executed should be accounted in "scheduling" time. More precisely:

- In starpu_profiling_worker_info::executing_time, the time spent executing kernels, normally equal to starpu_profiling_worker_info::all_executing_time.
- In starpu_profiling_worker_info::callback_time, the time spent executing application callbacks while not executing a task.
- In starpu_profiling_worker_info::waiting_time, the time spent waiting for data transfers while not executing a task or a callback.
- In starpu_profiling_worker_info::sleeping_time, the time spent during which there was no task to be executed and not executing a task or a callback or waiting for a data transfer, i.e. real lack of parallelism.
- In starpu_profiling_worker_info::scheduling_time, the time spent scheduling tasks while not executing a task or a callback or waiting for a data transfer to finish, and there are tasks to be scheduled.

This thus provides a split of the starpu_profiling_worker_info::total_time into various states. The difference between starpu_profiling_worker_info::total_time and the sum of this split is the remaining uncategorized overhead of the runtime.

Calling starpu_profiling_worker_get_info() resets the profiling information associated to a worker.

To easily display all this information, the environment variable STARPU_WORKER_STATS can be set to `1` (in addition to setting STARPU_PROFILING to 1). A summary will then be displayed at program termination. To display the summary in a file instead of the standard error stream, use the environment variable STARPU_WORKER_STATS_FILE.

```
Worker stats:
CUDA 0.0 (Tesla M2075 4.7 GiB 03:00.0)
        133 task(s)
        time split: total 3212.86 ms = executing: 1588.56 ms + callback: 2.95 ms + waiting: 5.34 ms + sleeping
        all time: executing: 1588.56 ms callback: 2.95 ms waiting: 22.83 ms sleeping: 1725.93 ms scheduling: 1
        286.388333 GFlop/s

CPU 0
        10 task(s)
        time split: total 3212.89 ms = executing: 2117.19 ms + callback: 0.23 ms + waiting: 0.01 ms + sleeping
        all time: executing: 2117.19 ms callback: 0.23 ms waiting: 0.01 ms sleeping: 1095.06 ms scheduling: 28
        22.029695 GFlop/s

CPU 1
        10 task(s)
        time split: total 3212.92 ms = executing: 2116.18 ms + callback: 0.17 ms + waiting: 0.01 ms + sleeping
        all time: executing: 2116.18 ms callback: 0.17 ms waiting: 0.01 ms sleeping: 1096.10 ms scheduling: 28
        22.029487 GFlop/s

CPU 2
        10 task(s)
        time split: total 3212.94 ms = executing: 2116.08 ms + callback: 0.18 ms + waiting: 0.01 ms + sleeping
        all time: executing: 2116.08 ms callback: 0.18 ms waiting: 0.01 ms sleeping: 1096.21 ms scheduling: 28
        22.029343 GFlop/s


Global time split: total 12851.60 ms = executing: 7938.01 ms (61.77%) + callback: 3.53 ms (0.03%) + waiting: 5
```

The number of GFlops/s is available because the starpu_task::flops field of the tasks were filled (or STARPU_FLOPS used in starpu_task_insert()).

When an FxT trace is generated (see Generating Traces With FxT), it is also possible to use the tool `starpu_workers_activity` (see Monitoring Activity) to generate a graphic showing the evolution of these values during the time, for the different workers.

22.1.5 Bus-related Feedback

The bus speed measured by StarPU can be displayed by using the tool `starpu_machine_display`, for instance:

```
StarPU has found:
        3 CUDA devices
                CUDA 0 (Tesla C2050 02:00.0)
                CUDA 1 (Tesla C2050 03:00.0)
                CUDA 2 (Tesla C2050 84:00.0)
from    to RAM          to CUDA 0       to CUDA 1       to CUDA 2
RAM     0.000000        5176.530428     5176.492994     5191.710722
CUDA 0  4523.732446     0.000000        2414.074751     2417.379201
CUDA 1  4523.718152     2414.078822     0.000000        2417.375119
CUDA 2  4534.229519     2417.069025     2417.060863     0.000000
```

Statistics about the data transfers which were performed and temporal average of bandwidth usage can be obtained by setting the environment variable STARPU_BUS_STATS to `1`; a summary will then be displayed at program termination. To display the summary in a file instead of the standard error stream, use the environment variable STARPU_BUS_STATS_FILE.

```
Data transfer stats:
        RAM 0 -> CUDA 0 319.92 MB       213.10 MB/s     (transfers : 91 - avg 3.52 MB)
        CUDA 0 -> RAM 0 214.45 MB       142.85 MB/s     (transfers : 61 - avg 3.52 MB)
        RAM 0 -> CUDA 1 302.34 MB       201.39 MB/s     (transfers : 86 - avg 3.52 MB)
        CUDA 1 -> RAM 0 133.59 MB       88.99 MB/s      (transfers : 38 - avg 3.52 MB)
        CUDA 0 -> CUDA 1        144.14 MB       96.01 MB/s      (transfers : 41 - avg 3.52 MB)
        CUDA 1 -> CUDA 0        130.08 MB       86.64 MB/s      (transfers : 37 - avg 3.52 MB)
        RAM 0 -> CUDA 2 312.89 MB       208.42 MB/s     (transfers : 89 - avg 3.52 MB)
        CUDA 2 -> RAM 0 133.59 MB       88.99 MB/s      (transfers : 38 - avg 3.52 MB)
        CUDA 0 -> CUDA 2        151.17 MB       100.69 MB/s     (transfers : 43 - avg 3.52 MB)
        CUDA 2 -> CUDA 0        105.47 MB       70.25 MB/s      (transfers : 30 - avg 3.52 MB)
        CUDA 1 -> CUDA 2        175.78 MB       117.09 MB/s     (transfers : 50 - avg 3.52 MB)
        CUDA 2 -> CUDA 1        203.91 MB       135.82 MB/s     (transfers : 58 - avg 3.52 MB)
Total transfers: 2.27 GB
```

22.1.6 MPI-related Feedback

Statistics about the data transfers which were performed over MPI can be obtained by setting the environment variable STARPU_MPI_STATS to `1`; a summary will then be displayed at program termination:

```
[starpu_comm_stats][1] TOTAL:   456.000000 B    0.000435 MB      0.000188 B/s    0.000000 MB/s
[starpu_comm_stats][1:0]        456.000000 B    0.000435 MB      0.000188 B/s    0.000000 MB/s
```

```
[starpu_comm_stats][0] TOTAL:   456.000000 B    0.000435 MB      0.000188 B/s     0.000000 MB/s
[starpu_comm_stats][0:1]        456.000000 B    0.000435 MB      0.000188 B/s     0.000000 MB/s
```

These statistics can be plotted as heatmaps using StarPU tool `starpu_mpi_comm_matrix.py` (see Debugging MPI).

22.2 Task And Worker Profiling

A full example showing how to use the profiling API is available in the StarPU sources in the directory `examples/profiling/`.

```
struct starpu_task *task = starpu_task_create();
task->cl = &cl;
task->synchronous = 1;
/* We will destroy the task structure by hand so that we can
* query the profiling info before the task is destroyed.  */
task->destroy = 0;
/* Submit and wait for completion (since synchronous was set to 1) */
starpu_task_submit(task);
/* The task is finished, get profiling information */
struct starpu_profiling_task_info *info = task->profiling_info;
/* How much time did it take before the task started ?  */
double delay += starpu_timing_timespec_delay_us(&info->submit_time, &info->start_time);
/* How long was the task execution ?  */
double length += starpu_timing_timespec_delay_us(&info->start_time, &info->end_time);
/* We no longer need the task structure */
starpu_task_destroy(task);
/* Display the occupancy of all workers during the test */
int worker;
for (worker = 0; worker < starpu_worker_get_count(); worker++)
{
        struct starpu_profiling_worker_info worker_info;
        int ret = starpu_profiling_worker_get_info(worker, &worker_info);
        STARPU_ASSERT(!ret);
        double total_time = starpu_timing_timespec_to_us(&worker_info.total_time);
        double executing_time = starpu_timing_timespec_to_us(&worker_info.executing_time);
        double sleeping_time = starpu_timing_timespec_to_us(&worker_info.sleeping_time);
        double overhead_time = total_time - executing_time - sleeping_time;
        float executing_ratio = 100.0*executing_time/total_time;
        float sleeping_ratio = 100.0*sleeping_time/total_time;
        float overhead_ratio = 100.0 - executing_ratio - sleeping_ratio;
        char workername[128];
        starpu_worker_get_name(worker, workername, 128);
        fprintf(stderr, "Worker %s:\n", workername);
        fprintf(stderr, "\ttotal time:  %.2lf ms\n", total_time*1e-3);
        fprintf(stderr, "\texec time:  %.2lf ms (%.2f %%)\n", executing_time*1e-3, executing_ratio);
        fprintf(stderr, "\tblocked time:  %.2lf ms (%.2f %%)\n", sleeping_time*1e-3, sleeping_ratio);
        fprintf(stderr, "\toverhead time:  %.2lf ms (%.2f %%)\n", overhead_time*1e-3, overhead_ratio);
}
```

22.3 Performance Model Example

To achieve good scheduling, StarPU scheduling policies need to be able to estimate in advance the duration of a task. This is done by giving to codelets a performance model, by defining a structure starpu_perfmodel and providing its address in the field starpu_codelet::model. The fields starpu_perfmodel::symbol and starpu_perfmodel::type are mandatory, to give a name to the model, and the type of the model, since there are several kinds of performance models. Then starpu_task_get_model_name() can be called to retrieve the name of the performance model associated with a task. For compatibility, make sure to initialize the whole structure to zero, either by using explicit memset(), or by letting the compiler implicitly do it as examplified below.

- Measured at runtime (model type STARPU_HISTORY_BASED). This assumes that for a given set of data input/output sizes, the performance will always be about the same. This is very true for regular kernels on GPUs for instance (<0.1% error), and just a bit less true on CPUs (~=1% error). This also assumes that there are few different sets of data input/output sizes. StarPU will then keep record of the average time of previous executions on the various processing units, and use it as an estimation. History is done per task size, by using a hash of the input and output sizes as an index. It will also save it in `$STARPU_HOME/.starpu/sampling/codelets` for further executions, and can be observed by using the tool `starpu_perfmodel_display`, or drawn by using the tool `starpu_perfmodel_plot` (Performance Model Calibration). The models are indexed by machine name. To share the models between machines (e.g. for a homogeneous cluster), use `export STARPU_HOSTNAME=some_global_name`. Measurements are only done when using a task scheduler which makes use of it, such as `dmda`. Measurements can also be provided explicitly by the application, by using the function starpu_perfmodel_update_history(). An example is in the file `tests/perfmodels/feed.c`.

The following is a small code example.

If e.g. the code is recompiled with other compilation options, or several variants of the code are used, the `symbol` string should be changed to reflect that, in order to recalibrate a new model from zero. The `symbol` string can even be constructed dynamically at execution time, as long as this is done before submitting any task using it.

```
static struct starpu_perfmodel mult_perf_model =
{
    .type = STARPU_HISTORY_BASED,
    .symbol = "mult_perf_model"
};
struct starpu_codelet cl =
{
    .cpu_funcs = { cpu_mult },
    .cpu_funcs_name = { "cpu_mult" },
    .nbuffers = 3,
    .modes = { STARPU_R, STARPU_R, STARPU_W },
    /* for the scheduling policy to be able to use performance models */
    .model = &mult_perf_model
};
```

- Measured at runtime and refined by regression (model types STARPU_REGRESSION_BASED and STARPU_NL_REGRESSION_BASED). This still assumes performance regularity, but works with various data input sizes, by applying regression over observed execution times. STARPU_REGRESSION_BASED uses an `a*n^b` regression form, STARPU_NL_REGRESSION_BASED uses an `a*n^b+c` (more precise than STARPU_REGRESSION_BASED, but costs a lot more to compute).

  For instance, `tests/perfmodels/regression_based.c` uses a regression-based performance model for the function `memset()`.

  Of course, the application has to issue tasks with varying size so that the regression can be computed. StarPU will not trust the regression unless there is at least 10% difference between the minimum and maximum observed input size. It can be useful to set the environment variable STARPU_CALIBRATE to `1` and run the application on varying input sizes with STARPU_SCHED set to `dmda` scheduler, to feed the performance model for a variety of inputs. The application can also provide the measurements explicitly by using the function starpu_perfmodel_update_history(). The tools `starpu_perfmodel_display` and `starpu_perfmodel_plot` can be used to observe how much the performance model is calibrated (Performance Model Calibration); when their output looks good, STARPU_CALIBRATE can be reset to `0` to let StarPU use the resulting performance model without recording new measures, and STARPU_SCHED can be set to `dmda` to benefit from the performance models. If the data input sizes vary a lot, it is really important to set STARPU_CALIBRATE to `0`, otherwise StarPU will continue adding the measures, and result with a very big performance model, which will take time a lot of time to load and save.

  For non-linear regression, since computing it is quite expensive, it is only done at termination of the application. This means that the first execution of the application will use only history-based performance model to perform scheduling, without using regression.

- Another type of model is STARPU_MULTIPLE_REGRESSION_BASED, which is based on multiple linear regression. In this model, users define both the relevant parameters and the equation for computing the task duration.

  $$T_{kernel} = a + b(M^{\alpha_1} * N^{\beta_1} * K^{\gamma_1}) + c(M^{\alpha_2} * N^{\beta_2} * K^{\gamma_2}) + ...$$

  M, N, K are the parameters of the task, added at the task creation. These need to be extracted by the `cl_perf_func` function, which should be defined by users. α, β, γ are the exponents defined by users in `model->combinations` table. Finally, coefficients a, b, c are computed automatically by the StarPU at the end of the execution, using least squares method of the `dgels_` LAPACK function.

  `examples/mlr/mlr.c` example provides more details on the usage of STARPU_MULTIPLE_REGRESSION_BASED models. The --enable-mlr configure option needs to be set to calibrate the model.

  Coefficients computation is done at the end of the execution, and the results are stored in standard codelet perfmodel files. Additional files containing the duration of tasks together with the value of each parameter are stored in `.starpu/sampling/codelets/tmp/` directory. These files are reused when STARPU_CALIBRATE environment variable is set to `1`, to recompute coefficients based on the current, but also on the previous executions. By default, StarPU uses a lightweight dgels implementation, but the --enable-mlr-system-blas configure option can be used to make StarPU use a system-provided dgels BLAS.

Additionally, when multiple linear regression models are not enabled through --enable-mlr or when the `model->combinations` are not defined, StarPU will still write output files into `.starpu/sampling/codelets/tmp/` to allow performing an analysis. This analysis typically aims at finding the most appropriate equation for the codelet and `tools/starpu_mlr_analysis` script provides an example of how to perform such study.

- Provided as an estimation from the application itself (model type STARPU_COMMON and field starpu_perfmodel::cost_function), see for instance `examples/common/blas_model.h` and `examples/common/blas_model.c`.
- Provided explicitly by the application (model type STARPU_PER_ARCH): either field starpu_perfmodel::arch_cost_function, or the fields `.per_arch[arch][nimpl].cost_function` have to be filled with pointers to functions which return the expected duration of the task in micro-seconds, one per architecture, see for instance `tests/datawizard/locality.c`
- Provided explicitly by the application (model type STARPU_PER_WORKER) similarly with the starpu_perfmodel::worker_cost_function field.

For STARPU_HISTORY_BASED, STARPU_REGRESSION_BASED, and STARPU_NL_REGRESSION_BASED, the dimensions of task data (both input and output) are used as an index by default. STARPU_HISTORY_BASED uses a CRC hash of the dimensions as an index to distinguish histories, and STARPU_REGRESSION_BASED and STARPU_NL_REGRESSION_BASED use the total size as an index for the regression. (Data marked with STARPU_NOFOOTPRINT are not taken into account).

The starpu_perfmodel::size_base and starpu_perfmodel::footprint fields however permit the application to override that, when for instance some of the data do not matter for task cost (e.g. mere reference table), or when using sparse structures (in which case it is the number of non-zeros which matter), or when there is some hidden parameter such as the number of iterations, or when the application actually has a very good idea of the complexity of the algorithm, and just not the speed of the processor, etc. The example in the directory `examples/pi` uses this to include the number of iterations in the base size. starpu_perfmodel::size_base should be used when the variance of the actual performance is known (i.e. bigger return value is longer execution time), and thus particularly useful for STARPU_REGRESSION_BASED or STARPU_NL_REGRESSION_BASED. starpu_perfmodel::footprint can be used when the variance of the actual performance is unknown (irregular performance behavior, etc.), and thus only useful for STARPU_HISTORY_BASED. starpu_task_data_footprint() can be used as a base and combined with other parameters through starpu_hash_crc32c_be() for instance.

StarPU will automatically determine when the performance model is calibrated, or rather, it will assume the performance model is calibrated until the application submits a task for which the performance can not be predicted. For STARPU_HISTORY_BASED, StarPU will require 10 (STARPU_CALIBRATE_MINIMUM) measurements for a given size before estimating that an average can be taken as estimation for further executions with the same size. For STARPU_REGRESSION_BASED and STARPU_NL_REGRESSION_BASED, StarPU will require 10 (STARPU_CALIBRATE_MINIMUM) measurements, and that the minimum measured data size is smaller than 90% of the maximum measured data size (i.e. the measurement interval is large enough for a regression to have a meaning). Calibration can also be forced by setting the STARPU_CALIBRATE environment variable to `1`, or even reset by setting it to `2`.

How to use schedulers which can benefit from such performance model is explained in Task Scheduling Policies.

The same can be done for task energy consumption estimation, by setting the field starpu_codelet::energy_model the same way as the field starpu_codelet::model. Note: for now, the application has to give to the energy consumption performance model a name which is different from the execution time performance model.

The application can request time estimations from the StarPU performance models by filling a task structure as usual without actually submitting it. The data handles can be created by calling any of the functions `starpu_*_data_register` with a `NULL` pointer and `-1` node and the desired data sizes, and need to be unregistered as usual. The functions starpu_task_expected_length() and starpu_task_expected_energy() can then be called to get an estimation of the task cost on a given arch. starpu_task_footprint() can also be used to get the footprint used for indexing history-based performance models. starpu_task_destroy() needs to be called to destroy the dummy task afterwards. See `tests/perfmodels/regression_based.c` for an example.

The application can also request an on-the-fly XML report of the performance model, by calling starpu_perfmodel_dump_xml() to print the report to a `FILE*`.

22.4 Performance Monitoring Counters

This section presents the StarPU performance monitoring framework. It summarizes the objectives of the framework. It then introduces the entities involved in the framework. It presents the API of the framework, as well as some implementation details. It exposes the typical sequence of operations to plug an external tool to monitor a performance counter of StarPU.

22.4.1 Objectives

The objectives of this framework are to let external tools interface with StarPU to collect various performance metrics at runtime, in a generic, safe, extensible way. For that, it enables such tools to discover the available performance metrics in a particular StarPU build, as well as the type of each performance counter value. It lets these tools build sets of performance counters to monitor, and then register listener callbacks to collect the measurement samples of these sets of performance counters at runtime.

22.4.2 Entities

The performance monitoring framework is built on a series of concepts and items, organized consistently. The corresponding C language objects should be considered opaque by external tools, and should only be manipulated through proper function calls and accessors.

22.4.2.1 Performance Counter

The performance counter entity is the fundamental object of the framework, representing one piece of performance metrics, such as for instance the total number of tasks submitted so far, that is exported by StarPU and can be collected through the framework at runtime. A performance counter has a type and belongs to a scope. A performance counter is designated by a unique name and unique ID integer. We can start or stop collecting performance counter values by using starpu_perf_counter_collection_start() and starpu_perf_counter_collection_stop().

22.4.2.2 Performance Counter Type

A performance counter has a type. A type is designated by a unique name and unique ID number. Currently, supported types include:

Type Name	Type Definition
"int32"	32-bit signed integers
"int64"	64-bit signed integers
"float"	32-bit single-precision floating point
"double"	64-bit double-precision floating point

22.4.2.3 Performance Counter Scope

A performance counter belongs to a scope. The scope of a counter defines the context considered for computing the corresponding performance counter. A scope is designated with a unique name and unique ID number. Currently, defined scopes include:

Scope Name	Scope Definition
"global"	Counter is global to the StarPU instance
"per_worker"	Counter is within the scope of a thread worker
"per_codelet"	Counter is within the scope of a task codelet

22.4.2.4 Performance Counter Set

A performance counter set is a subset of the performance counters belonging to the same scope. Each counter of the scope can be in the enabled or disabled state in a performance counter set. A performance counter set enables a performance monitoring tool to indicate the set of counters to be collected for a particular listener callback.

22.4.2.5 Performance Counter Sample

A performance counter sample corresponds to one sample of collected measurement values of a performance counter set. Only the values corresponding to enabled counters in the sample's counter set should be observed by the listener callback. Whether the sample contains valid values for counters disabled in the set is unspecified.

22.4.2.6 Performance Counter Listener

A performance counter listener is a callback function registered by some external tool to monitor a set of performance counters in a particular scope. It is called each time a new performance counter sample is ready to be observed. The sample object should not be accessed outside the callback.

22.4.2.7 Application Programming Interface

The API of the performance monitoring framework is defined in the starpu_perf_monitoring.h public header file of StarPU. This header file is automatically included with starpu.h. An example of use of the routines is given in Sequence of operations.

22.4.3 Implementation Details

22.4.3.1 Performance Counter Registration

Each module of StarPU can export performance counters. In order to do so, modules that need to export some counters define a registration function that is called at StarPU initialization time. This function is responsible for calling the "_starpu_perf_counter_register()" function once for each counter it exports, to let the framework know about the list of counters managed by the module. It also registers performance sample updater callbacks for the module, one for each scope for which it exports counters.

22.4.3.2 Performance Sample Updaters

The updater callback for a module and scope combination is internally called every time a sample for a set of performance counter must be updated. Thus, the updated callback is responsible for filling the sample's selected counters with the counter values found at the time of the call. Global updaters are currently called at task submission time, as well as any blocking tasks management function of the StarPU API, such as starpu_task_wait_for_all(), which waits for the completion of all tasks submitted up to this point. Per-worker updaters are currently called at the level of StarPU's drivers, that is, the modules in charge of task execution of hardware-specific worker threads. The actual calls occur in-between the execution of tasks. Per-codelet updaters are currently called both at task submission time, and at the level of StarPU's drivers together with the per-worker updaters.
A performance sample object is locked during the sample collection. The locking prevents the following issues:

- The listener of sample being changed during sample collection;
- The set of counters enabled for a sample being changed;
- Conflicting concurrent updates;
- Updates while the sample is being read by the listener.

The location of the updaters' calls is chosen to minimize the sequentialization effect of the locking, in order to limit the level of interference of the monitoring process. For Global updaters, the calls are performed only on the application thread(s) in charge of submitting tasks. Since, in most cases, only a single application thread submits tasks, the sequentialization effect is moderate. Per-worker updates are local to their worker, thus here again the sample lock is un-contented, unless the external monitoring tool frequently changes the set of enabled counters in the sample.

22.4.3.3 Counter operations

In practice, the sample updaters only take snapshots of the actual performance counters. The performance counters themselves are updated with ad-hoc procedures depending on each counter. Such procedures typically involve atomic operations. While operations such as atomic increments or decrements on integer values are readily available, this is not the case for more complex operations such as min/max for computing peak value counters (for instance in the global and per-codelet counters for peak number of submitted tasks and peak number of ready tasks waiting for execution), and this is also not the case for computations on floating point data (used for instance in computing cumulated execution time of tasks, either per worker or per codelet). The performance monitoring framework therefore supplies such missing routines, for the internal use of StarPU.

22.4.3.4 Runtime checks

The performance monitoring framework features a comprehensive set of runtime checks to verify that both StarPU and some external tool do not access a performance counter with the wrong typed routines, to quickly detect situations of mismatch that can result from the evolution of multiple pieces of software at distinct paces. Moreover, no StarPU data structure is accessed directly, either by the external code making use of the performance monitoring framework. The use of the C enum constants is optional; referring to values through constant strings is available when more robustness is desired. These runtime checks enable the framework to be extensible. Moreover, while the framework's counters currently are permanently compiled in, they could be made optional at compile time, for instance to suppress any overhead once the analysis and optimization process has been completed by the programmer. Thanks to the runtime discovery of available counters, the applicative code, or an intermediate layer such as skeleton layer acting on its behalf, would then be able to adapt to performance analysis builds versus optimized builds.

22.4.4 Exported Counters

22.4.4.1 Global Scope

Counter Name	Counter Definition
`starpu.task.g_total_submitted`	Total number of tasks submitted
`starpu.task.g_peak_submitted`	Maximum number of tasks submitted, waiting for dependencies resolution at any time
`starpu.task.g_peak_ready`	Maximum number of tasks ready for execution, waiting for an execution slot at any time

22.4.4.2 Per-worker Scope

Counter Name	Counter Definition
`starpu.task.w_total_executed`	Total number of tasks executed on a given worker
`starpu.task.w_cumul_execution_time`	Cumulated execution time of tasks executed on a given worker

22.4.4.3 Per-Codelet Scope

Counter Name	Counter Definition
`starpu.task.c_total_submitted`	Total number of submitted tasks for a given codelet
`starpu.task.c_peak_submitted`	Maximum number of submitted tasks for a given codelet waiting for dependencies resolution at any time
`starpu.task.c_peak_ready`	Maximum number of ready tasks for a given codelet waiting for an execution slot at any time
`starpu.task.c_total_executed`	Total number of executed tasks for a given codelet
`starpu.task.c_cumul_execution_time`	Cumulated execution time of tasks for a given codelet

22.4.5 Sequence of operations

This section presents a typical sequence of operations to interface an external tool with some StarPU performance counters. In this example, the counters monitored are the per-worker total number of executed tasks (`starpu.task.w_total_executed`) and the tasks' cumulated execution time (`starpu.task.w_cumul_execution_time`).

Step 0: Initialize StarPU

StarPU must first be initialized, by a call to starpu_init(), for performance counters to become available, since each module of StarPU registers the performance counters it exports during that initialization phase.

```
int ret = starpu_init(NULL);
```

Step 1: Allocate a counter set

A counter set has to be allocated on the per-worker scope. The per-worker scope id can be obtained by name, or with the pre-defined enum value starpu_perf_counter_scope_per_worker.

```
enum starpu_perf_counter_scope w_scope = starpu_perf_counter_scope_per_worker;
struct starpu_perf_counter_set *w_set = starpu_perf_counter_set_alloc(w_scope);
```

Step 2: Get the counter IDs Each performance counter has a unique ID used to refer to it in subsequent calls to the performance monitoring framework.

```
int id_w_total_executed = starpu_perf_counter_name_to_id(w_scope,
                                                     "starpu.task.w_total_executed");
int id_w_cumul_execution_time = starpu_perf_counter_name_to_id(w_scope,
                                               "starpu.task.w_cumul_execution_time");
```

Step 3: Enable the counters in the counter set

This step indicates which counters will be collected into performance monitoring samples for the listeners referring to this counter set.

```
starpu_perf_counter_set_enable_id(w_set, id_w_total_executed);
starpu_perf_counter_set_enable_id(w_set, id_w_cumul_execution_time);
```

Step 4: Write a listener callback

This callback will be triggered when a sample becomes available. Upon execution, it reads the values for the two counters from the sample and displays these values, for the sake of the example.

```
void w_listener_cb(struct starpu_perf_counter_listener *listener,
                   struct starpu_perf_counter_sample   *sample,
                   void *context)
{
  int32_t w_total_executed =
             starpu_perf_counter_sample_get_int32_value(sample, id_w_total_executed);
  double w_cumul_execution_time =
      starpu_perf_counter_sample_get_double_value(sample, id_w_cumul_execution_time);
  printf("worker[%d]:  w_total_executed = %d, w_cumul_execution_time = %lf\n",
                                 starpu_worker_get_id(),
                                 w_total_executed,
                                 w_cumul_execution_time);
}
```

Step 5: Initialize the listener

This step allocates the listener structure and prepares it to listen to the selected set of per-worker counters. However, it is not actually active until Step 6, once it is attached to one or more worker.

```
struct starpu_perf_counter_listener * w_listener =
                       starpu_perf_counter_listener_init(w_set, w_listener_cb, NULL);
```

Step 6: Set the listener on all workers This step actually makes the listener active, in this case on every StarPU worker thread.

```
starpu_perf_counter_set_all_per_worker_listeners(w_listener);
```

After this step, any task assigned to a worker will be counted in that worker selected performance counters, and reported to the listener.

22.5 Performance Steering Knobs

This section presents the StarPU performance steering framework. It summarizes the objectives of the framework. It introduces the entities involved in the framework, and then details the API, implementation and sequence of operations.

22.5.1 Objectives

The objectives of this framework are to let external tools interface with StarPU, observe, and act at runtime on actionable performance steering knobs exported by StarPU, in a generic, safe, extensible way. It defines an API to let such external tools discover the available performance steering knobs in a particular StarPU revision of build, as well as the type of each knob.

22.5.2 Entities

22.5.2.1 Performance Steering Knob

The performance steering knob entity designates one runtime-actionable knob exported by StarPU. It may represent some setting, or some constant used within StarPU for a given purpose. The value of the knob is typed, it can be obtained or modified with the appropriate getter/setter routine. The knob belongs to a scope. A performance steering knob is designated with a unique name and unique ID number.

22.5.2.2 Knob Type

A performance steering knob has a type. A type is designated by a unique name and unique ID number. Currently, supported types include:

Type Name	Type Definition
"int32"	32-bit signed integers
"int64"	64-bit signed integers
"float"	32-bit single precision floating point
"double"	64-bit double precision floating point

On/Off knobs are defined as "int32" type, with value 0 for Off and value !0 for On, unless otherwise specified.

22.5.2.3 Knob Scope

A performance steering knob belongs to a scope. The scope of a knob defines the context considered for computing the corresponding knob. A scope is designated with a unique name and unique ID number. Currently, defined scopes include:

Scope Name	Scope Definition
"global"	Knob is global to the StarPU instance
"per_worker"	Knob is within the scope of a thread worker
"per_scheduler"	Knob is within the scope of a scheduling policy instance

22.5.2.4 Knob Group

The notion of Performance Steering Knob Group is currently internal to StarPU. It defines a series of knobs that are handled by the same couple of setter/getter functions internally. A knob group belongs to a knob scope.

22.5.3 Application Programming Interface

The API is defined in the starpu_perf_steering.h public header file of StarPU. This header file is automatically included with starpu.h.

22.5.4 Implementation Details

While the APIs of the monitoring and the steering frameworks share a similar design philosophy, the internals are significantly different. Since the effect of the steering knobs varies widely, there is no global locking scheme in place shared for all knobs. Instead, each knob gets its own procedures to get the value of a setting, or change it. To prevent code duplication, some related knobs may share getter/setter routines as knob groups.

The steering framework does not involve callback routines. Knob get operations proceed immediately, except for the possible delay in getting access to the knob value. Knob set operations also proceed immediately, not counting the exclusive access time, though their action result may be observed with some latency, depending on the knob and on the current workload. For instance, acting on a per-worker `starpu.worker.w_enable_worker_knob` to disable a worker thread may be observed only after the corresponding worker's assigned task queue becomes empty, since its actual effect is to prevent additional tasks to be queued to the worker, and not to migrate already queued tasks to another worker. Such design choices aim at providing a compromise between offering some steering capabilities and keeping the cost of supporting such steering capabilities to an acceptable level.

The framework is designed to be easily extensible. At StarPU initialization time, the framework calls initialization functions if StarPU modules to initialize the set of knobs they export. Knob get/set accessors can be shared among multiple knobs in a knob group. Thus, exporting a new knob is basically a matter of declaring it at initialization time, by specifying its name and value type, and either add its handling to an existing getter/setter pair of accessors in a knob group, or create a new group. As the performance monitoring framework, the performance steering framework is currently permanently enabled, but could be made optional at compile-time to separate testing builds from production builds.

22.5.5 Exported Steering Knobs

22.5.5.1 Global Scope

Knob Name	Knob Definition
`starpu.global.g_calibrate_knob`	Enable/disable the calibration of performance models
`starpu.global.g_enable_catch_signal_knob`	Enable/disable the catching of UNIX signals

22.5.5.2 Per-worker Scope

Knob Name	Knob Definition
`starpu.worker.w_bind_to_pu_knob`	Change the processing unit to which a worker thread is bound
`starpu.worker.w_enable_worker_knob`	Disable/re-enable a worker thread to be selected for task execution

22.5.5.3 Per-Scheduler Scope

Knob Name	Knob Definition
`starpu.task.s_max_priority_cap_knob`	Set a capping maximum priority value for subsequently submitted tasks
`starpu.task.s_min_priority_cap_knob`	Set a capping minimum priority value for subsequently submitted tasks
`starpu.dmda.s_alpha_knob`	Scaling factor for the Alpha constant for Deque Model schedulers to alter the weight of the estimated task execution time
`starpu.dmda.s_beta_knob`	Scaling factor for the Beta constant for Deque Model schedulers to alter the weight of the estimated data transfer time for the task's input(s)
`starpu.dmda.s_gamma_knob`	Scaling factor for the Gamma constant for Deque Model schedulers to alter the weight of the estimated power consumption of the task
`starpu.dmda.s_idle_power_knob`	Scaling factor for the baseline Idle power consumption estimation of the corresponding processing unit

22.5.6 Sequence of operations

This section presents an example of a sequence of operations representing a typical use of the performance steering knobs exported by StarPU. In this example, a worker thread is temporarily barred from executing tasks. For that, the corresponding `starpu.worker.w_enable_worker_knob` of the worker, initially set to 1 (= enabled) is changed to 0 (= disabled).

Step 0: Initialize StarPU

StarPU must first be initialized, by a call to starpu_init(). Performance steering knobs only become available after this step, since each module of StarPU registers the knobs it exports during that initialization phase.

```
int ret = starpu_init(NULL);
```

Step 1: Get the knob ID

Each performance steering knob has a unique ID used to refer to it in subsequent calls to the performance steering framework. The knob belongs to the "per_worker" scope.

```
int w_scope = starpu_perf_knob_scope_name_to_id("per_worker");
int w_enable_id = starpu_perf_knob_name_to_id(w_scope, "starpu.worker.w_enable_worker_knob");
```

Step 2: Get the knob current value

This knob is an On/Off knob. Its value type is therefore a 32-bit integer, with value 0 for Off and value !0 for On. The getter functions for per-worker knobs expect the knob ID as first argument, and the worker ID as second argument. Here the getter call obtains the value of worker 5.

```
int32_t val = starpu_perf_knob_get_per_worker_int32_value(w_enable_id, 5);
```

Step 3: Set the knob current value

The setter functions for per-worker knobs expect the knob ID as first argument, the worker ID as second argument, and the new value as third argument. Here, the value for worker 5 is set to 0 to temporarily bar the worker thread from accepting new tasks for execution.

```
starpu_perf_knob_set_per_worker_int32_value(w_enable_id, 5, 0);
```

Subsequently, setting the value of the knob back to 1 enables the corresponding to accept new tasks for execution again.

```
starpu_perf_knob_set_per_worker_int32_value(w_enable_id, 5, 1);
```

Chapter 23

Offline Performance Tools

To get an idea of what is happening, a lot of performance feedback is available, detailed in this chapter. The various information should be checked for.

- What does the Gantt diagram look like? (see Creating a Gantt Diagram)
  - If it's mostly green (tasks running in the initial context) or context specific color prevailing, then the machine is properly utilized, and perhaps the codelets are just slow. Check their performance, see Performance Of Codelets.
  - If it's mostly purple (FetchingInput), tasks keep waiting for data transfers, do you perhaps have far more communication than computation? Did you properly use CUDA streams to make sure communication can be overlapped? Did you use data-locality aware schedulers to avoid transfers as much as possible?
  - If it's mostly red (Blocked), tasks keep waiting for dependencies, do you have enough parallelism? It might be a good idea to check what the DAG looks like (see Creating a DAG With Graphviz).
  - If only some workers are completely red (Blocked), for some reason the scheduler didn't assign tasks to them. Perhaps the performance model is bogus, check it (see Performance Of Codelets). Do all your codelets have a performance model? When some of them don't, the schedulers switches to a greedy algorithm which thus performs badly.

You can also use the Temanejo task debugger (see Using The Temanejo Task Debugger) to visualize the task graph more easily.

23.1 Generating Traces With FxT

StarPU can use the FxT library (see `https://savannah.nongnu.org/projects/fkt/`) to generate traces with a limited runtime overhead.
You can get a tarball from `http://download.savannah.gnu.org/releases/fkt/?C=M`
Compiling and installing the FxT library in the `$FXTDIR` path is done following the standard procedure:

```
$ ./configure --prefix=$FXTDIR
$ make
$ make install
```

In order to have StarPU to generate traces, StarPU needs to be configured again after installing FxT, and configuration show:

```
FxT trace enabled: yes
```

If `configure` does not find FxT automatically, it can be specified by hand with the option --with-fxt :

```
$ ./configure --with-fxt=$FXTDIR
```

Or you can simply point the `PKG_CONFIG_PATH` environment variable to `$FXTDIR/lib/pkgconfig`
When STARPU_FXT_TRACE is set to 1, a trace is generated when StarPU is terminated by calling starpu_shutdown(). The trace is a binary file whose name has the form `prof_file_XXX_YYY` where `XXX` is the username, and `YYY` is the MPI id of the process that used StarPU (or 0 when running a sequential program).

One can change the name of the file by setting the environment variable STARPU_FXT_SUFFIX, its contents will be used instead of `prof_file_XXX`. This file is saved in the `/tmp/` directory by default, or by the directory specified by the environment variable STARPU_FXT_PREFIX.
The additional `configure` option --enable-fxt-lock can be used to generate trace events which describes the lock's behavior during the execution. It is however very heavy and should not be used unless debugging StarPU's internal locking.
When the FxT trace file `prof_file_something` has been generated, it is possible to generate different trace formats by calling:

```
$ starpu_fxt_tool -i /tmp/prof_file_something
```

Or alternatively, setting the environment variable STARPU_GENERATE_TRACE to `1` before application execution will make StarPU automatically generate all traces at application shutdown. Note that if the environment variable STARPU_FXT_PREFIX is set, files will be generated in the given directory.
One can also set the environment variable STARPU_GENERATE_TRACE_OPTIONS to specify options, see `starpu_fxt_tool -help`, for example:

```
$ export STARPU_GENERATE_TRACE=1
$ export STARPU_GENERATE_TRACE_OPTIONS="-no-acquire"
```

When running an MPI application, STARPU_GENERATE_TRACE will not work as expected (each node will try to generate trace files, thus mixing outputs...), you have to collect the trace files from the MPI nodes, and specify them all on the command `starpu_fxt_tool`, for instance:

```
$ starpu_fxt_tool -i /tmp/prof_file_something*
```

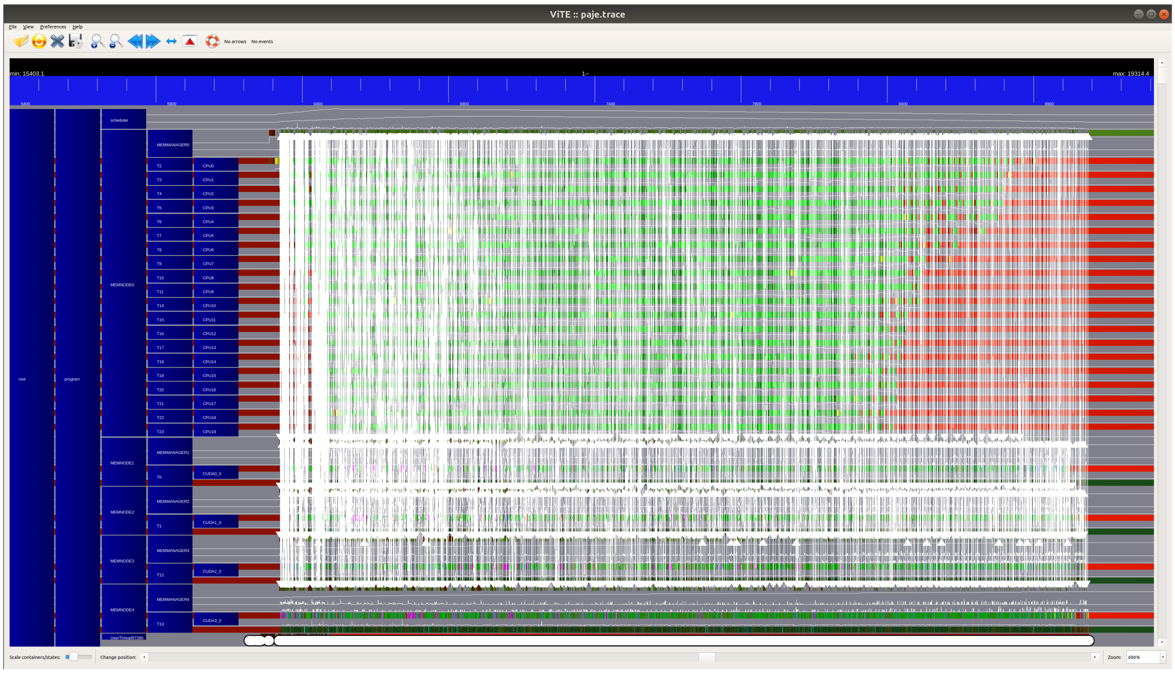
By default, the generated trace contains all information. To reduce the trace size, various `-no-foo` options can be passed to `starpu_fxt_tool`, see `starpu_fxt_tool -help` .

23.1.1 Creating a Gantt Diagram

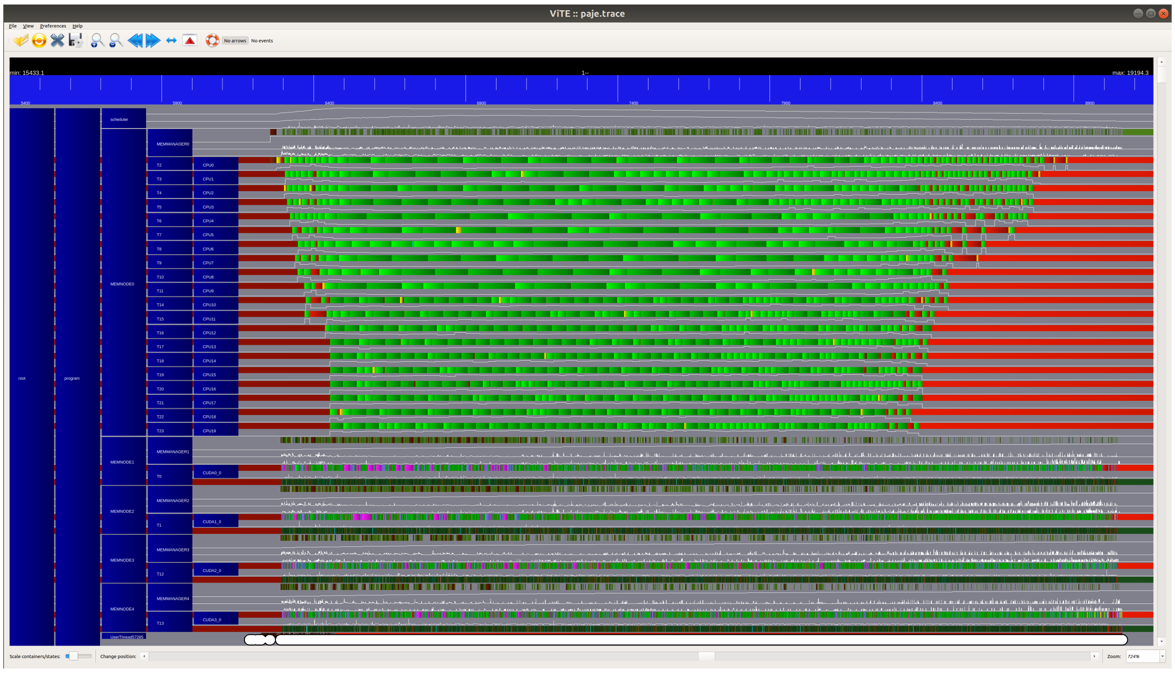
One of the generated files is a trace in the Paje format. The file, located in the current directory, is named `paje.trace`. It can be viewed with ViTE (`https://solverstack.gitlabpages.inria.fr/vite/`) a trace visualizing open-source tool. To open the file `paje.trace` with ViTE, use the following command:

```
$ vite paje.trace
```

Once the file is opened in ViTE interface, we will see the figure as shown below:



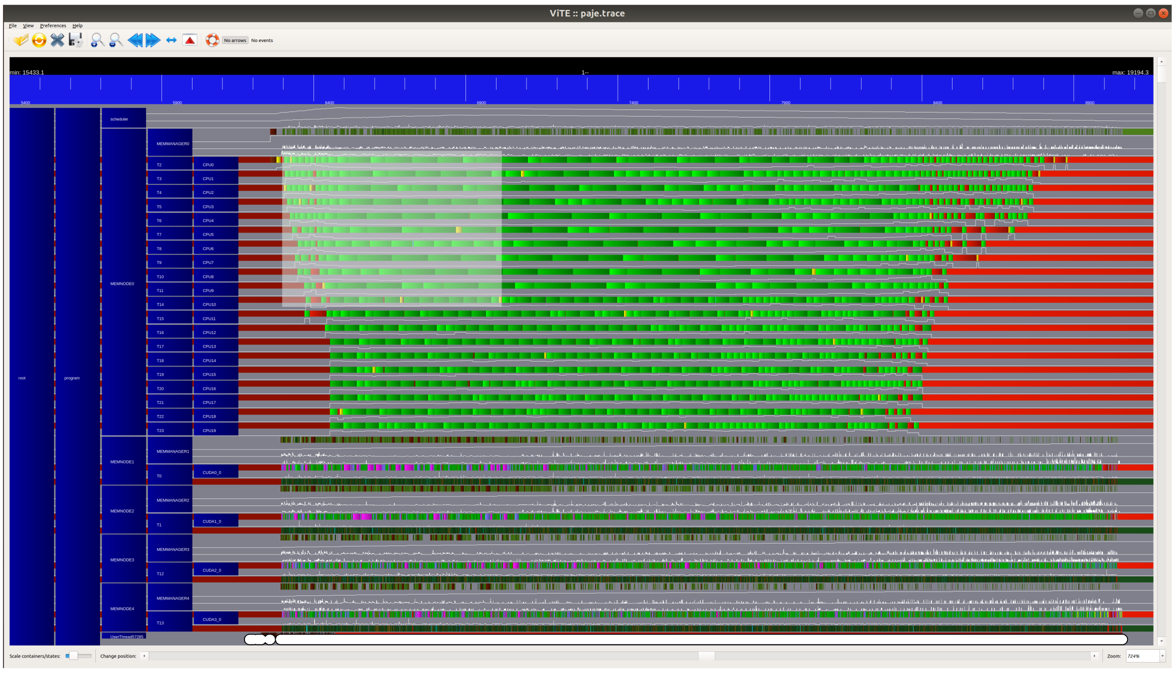
We can then click the "No arrows" button in task bar of ViTE interface, to better observe the Gantt diagram that illustrates the start and end dates of the different tasks or activities of a program.



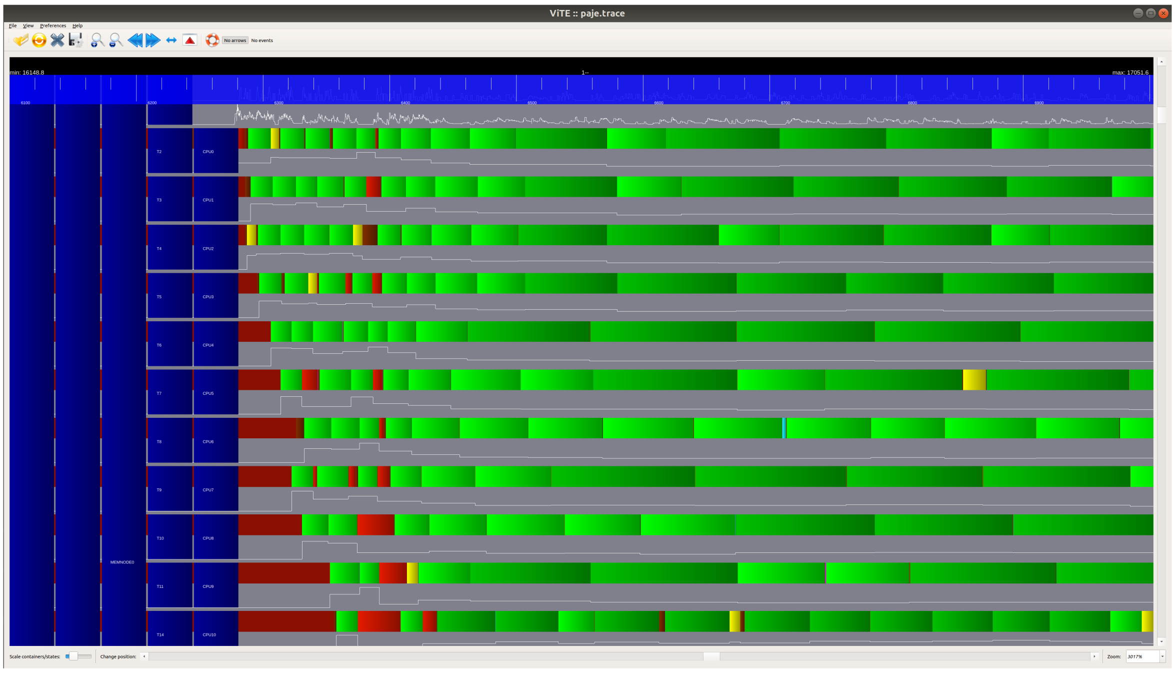
In the Gantt diagram, the bar types such as devices (CPU or GPU) are displayed on the left side. Each task is represented by a horizontal rectangle that spans the duration of the task. The rectangles are arranged along a timeline axis, which is shown at the top of the Gantt diagram and represents the overall duration of the program in milliseconds. The position of the bar along the timeline shows when the task begins and ends. We can see some long red bars at the beginning and end of the entire timeline, which represent that the unit is idle. There are no tasks at these moments, and workers are waiting or in a sleeping state.

23.1.1.1 Zooming in Gantt Diagram

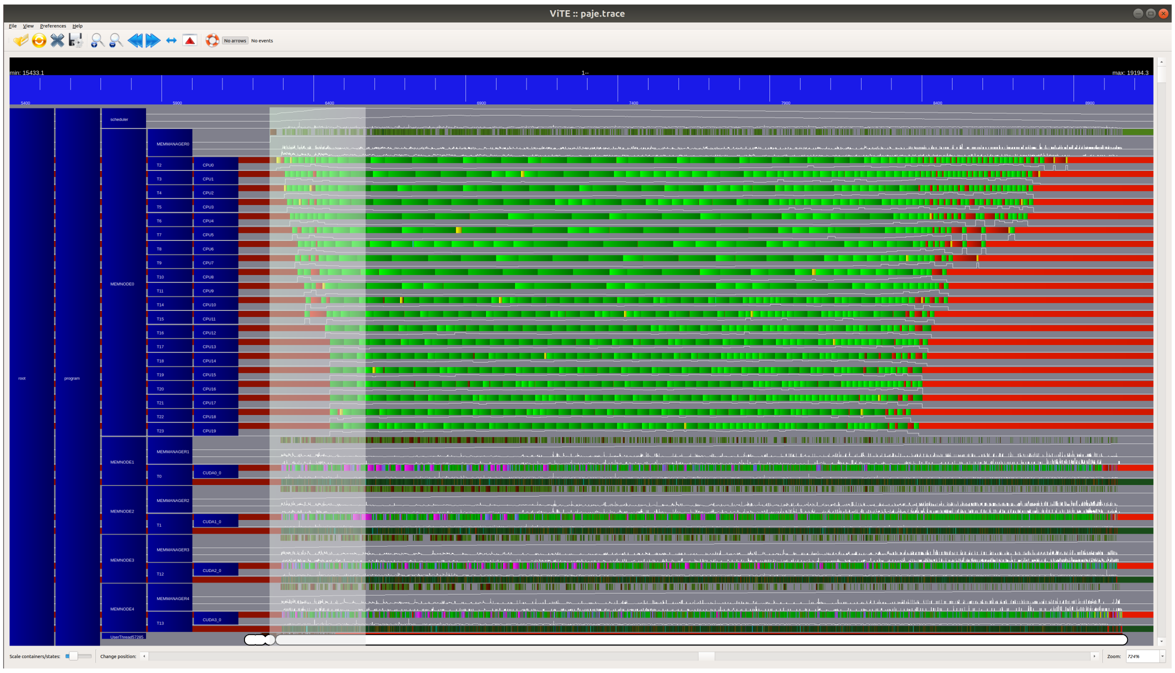
Then as shown in the following figure, press and hold the left mouse button to select the area you want to zoom in on. Release the button to view the selected area, and we can repeat the zoom action multiple times.



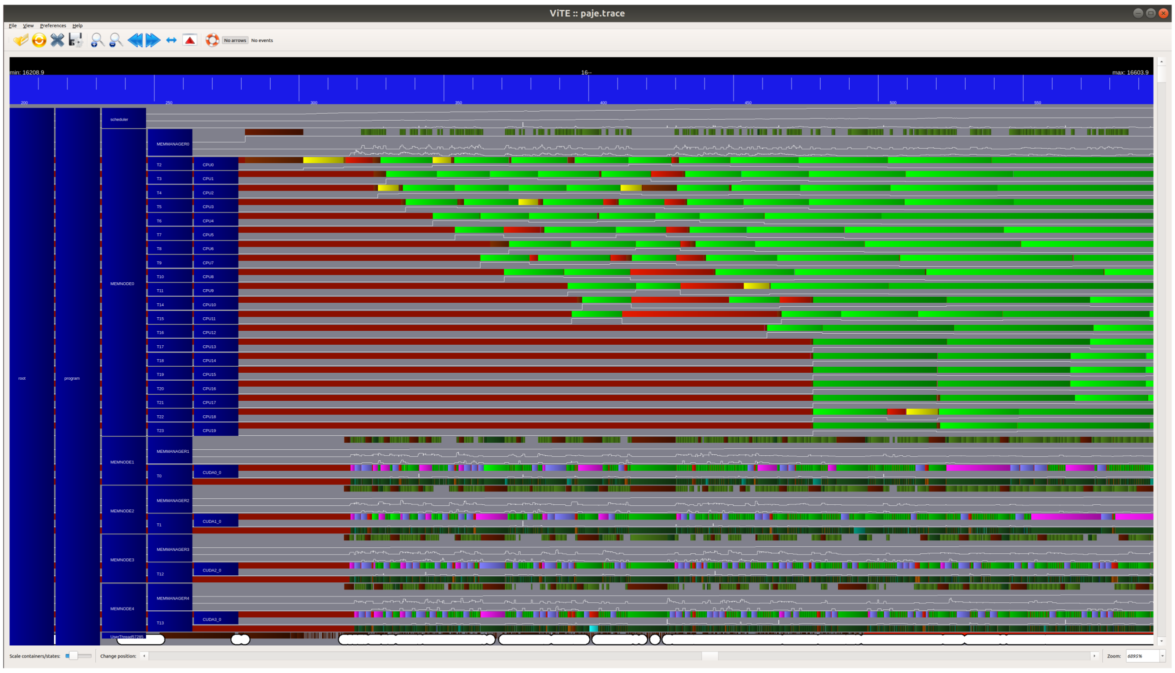
This zoom result is:



Right-clicking anywhere on the Gantt diagram restores the previous zoom view.
One can press and hold the left mouse button inside the top blue bar to select horizontally, which will horizontally zoom in on all Gantt diagrams within the selected time range.

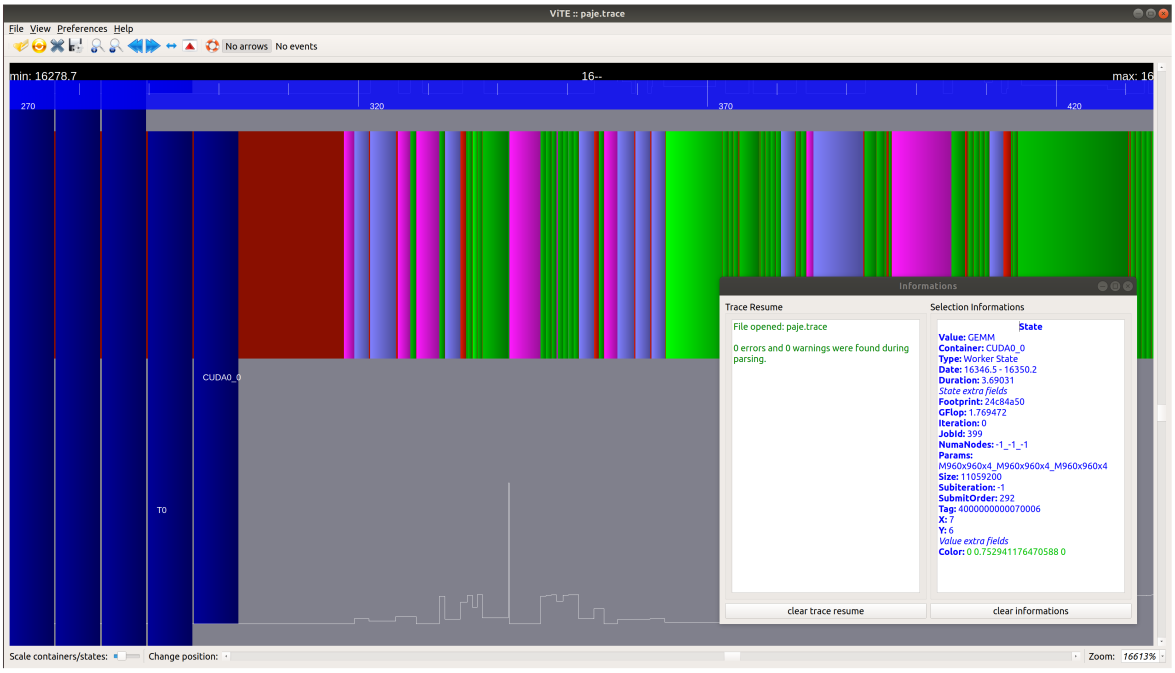


This zoom result is:



23.1.1.2 Colors in Gantt Diagram

After zooming in, we can observe numerous blocks of varying colors, each block representing a task. Blocks of diverse colors signify different types of tasks. When we double-click on any block, a pop-up window will show related status about that task, such as its type and which worker (CPU/GPU) it belongs to, etc.



File opened: paje.trace

0 errors and 0 warnings were found during parsing.

State

Value: GEMM
Container: CUDA0_0
Type: Worker State
Date: 16346.5 - 16350.2
Duration: 3.69031
State extra fields
Footprint: 24c84a50
GFlop: 1.769472
Iteration: 0
JobId: 399
NumaNodes: -1_-1_-1
Params: M960x960x4_M960x960x4_M960x960x4
Size: 11059200
Subiteration: -1
SubmitOrder: 292
Tag: 4000000000070006
X: 7
Y: 6
Value extra fields
Color: 0 0.752941176470588 0

The state information displayed in the pop-up window can be:

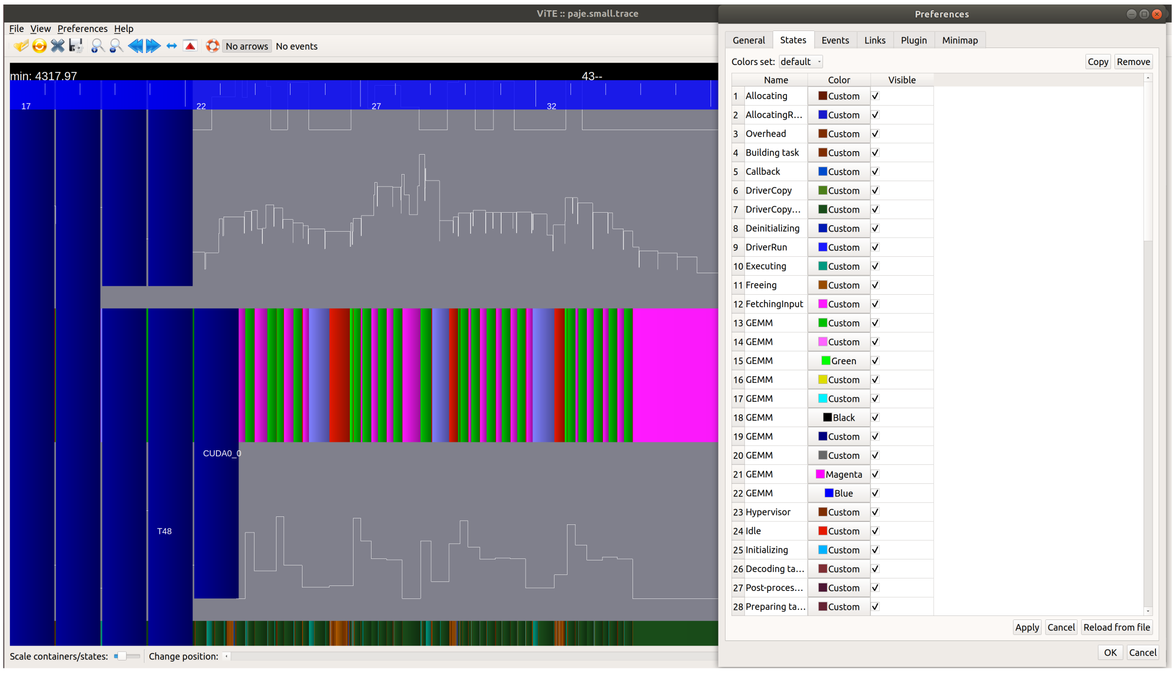
- **Value**: refers to a type of task, which can be assigned as a task name (instead of the default `unknown`) by filling the optional starpu_codelet::name, or assigning it a performance model. The name can also be set with the field starpu_task::name or by using STARPU_NAME when calling starpu_task_insert()
- **Container**: refers to a specific worker where the computation was performed, could be CPU or CUDA
- **Type**: indicates the type of this block, most often "Worker State"

- **Date**: represents a range of dates during which the computation was performed
- **Duration**: represents the duration of the computation
- **Footprint**: provides the data footprint of the task (used as indexing base for performance models)
- **GFlop**: represents the number of Gflop performed during the computation, as set in starpu_task::flops.
- **Iteration**: refers to the iteration number of the computation, as set by starpu_iteration_push() at the beginning of submission loops and starpu_iteration_pop() at the end of submission loops
- **JobId**: represents a unique identifier for the specific task, as returned by starpu_task_get_job_id()
- **NumaNodes**: refers to the NUMA node where the data is stored, the environment variable STARPU_FXT_EVENTS needs to contain `TASK_VERBOSE_EXTRA`, otherwise it will be -1
- **Params**: represents parameters or input/output types and sizes, possibly indicating the dimensions of the matrices
- **Size**: represents the size of the data being operated on in bytes
- **Subiteration**: represents a sub-iteration number if the computation was part of a larger iteration or loop, as set by starpu_iteration_push()
- **SubmitOrder**: represents the order in which the task was submitted by the application
- **Tag**: represents a unique identifier for the task, which can be set either through starpu_task::tag_id or by using STARPU_TAG or STARPU_TAG_ONLY when calling starpu_task_insert()
- **X**: represents an X-coordinate index of the first data written by the task, which was set by starpu_data_set_coordinates() or starpu_data_set_coordinates_array() function. We can also get the coordinates of the data with starpu_data_get_coordinates_array() function
- **Y**: represents an Y-coordinate index of the first data written by the task, which was set by starpu_data_set_coordinates() or starpu_data_set_coordinates_array() function. We can also get the coordinates of the data with starpu_data_get_coordinates_array() function
- **Color**: represents the color RGB value associated with the task. Tasks are by default shown in green. To use a different color for every type of task, we can specify the option `-c` to `starpu_fxt_tool` or in STARPU_GENERATE_TRACE_OPTIONS. Tasks can also be given a specific color by setting the field starpu_codelet::color or the starpu_task::color. When we call starpu_task_insert(), we can use STARPU_TASK_COLOR to set the color. Colors are expressed with the following format `0xRRGGBB` (e.g. `0xFF0000` for red). See `basic_examples/task_insert_color` for examples on how to assign colors

In the shown figure, the set of color as following:

- Dark green represents GEMM
- Light green represents SYRK
- Blue represents TRSM
- Red indicates that the unit is idle, there are no tasks at the moment, it is currently waiting or in a sleeping state
- Magenta represents FetchingInput

To modify the colors in Vite interface, select "Preferences" then "Settings" in the options bar, and then choose the "States" tab in the newly opened window to select different colors for different operations, as shown in the figure below. One has to click the reload button at the top left to reload the trace with the new colors.

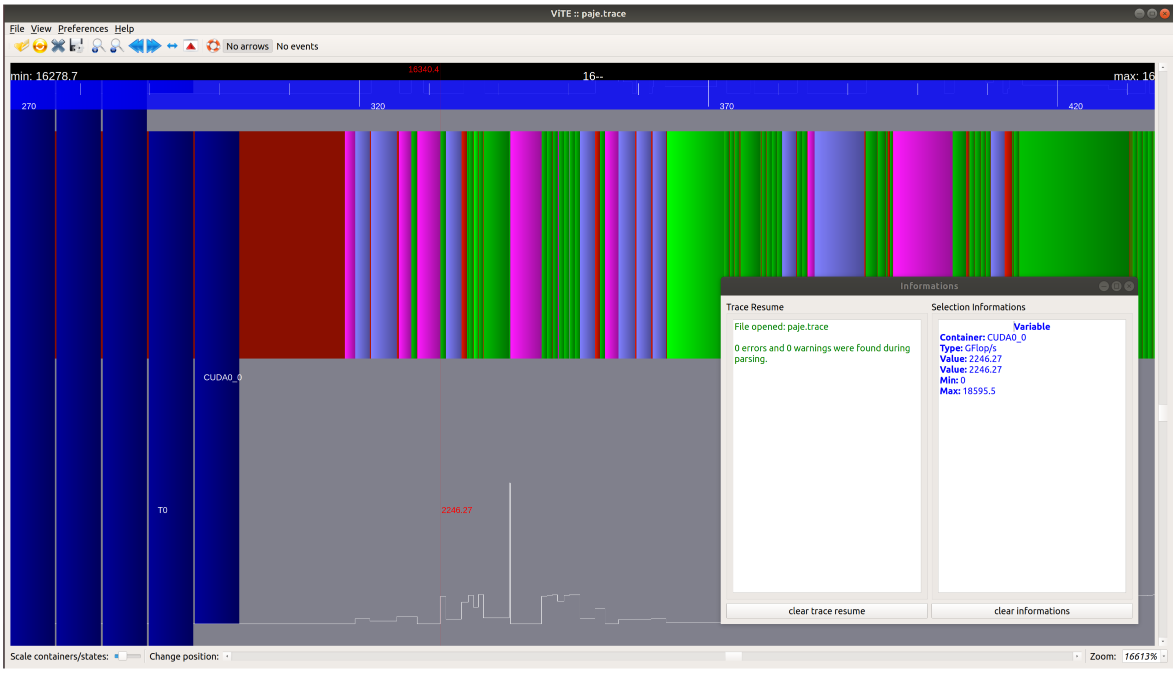


Colors set: default

	Name	Color	Visible
1	Allocating	Custom	✓
2	AllocatingR…	Custom	✓
3	Overhead	Custom	✓
4	Building task	Custom	✓
5	Callback	Custom	✓
6	DriverCopy	Custom	✓
7	DriverCopy…	Custom	✓
8	Deinitializing	Custom	✓
9	DriverRun	Custom	✓
10	Executing	Custom	✓
11	Freeing	Custom	✓
12	FetchingInput	Custom	✓
13	GEMM	Custom	✓
14	GEMM	Custom	✓
15	GEMM	Green	✓
16	GEMM	Custom	✓
17	GEMM	Custom	✓
18	GEMM	Black	✓
19	GEMM	Custom	✓
20	GEMM	Custom	✓
21	GEMM	Magenta	✓
22	GEMM	Blue	✓
23	Hypervisor	Custom	✓
24	Idle	Custom	✓
25	Initializing	Custom	✓
26	Decoding ta…	Custom	✓
27	Post-proces…	Custom	✓
28	Preparing ta…	Custom	✓

23.1.1.3 Curves in Gantt Diagram

We can see that there is a curve below task blocks, which represents the corresponding `GFlop/s`. Double-clicking near the curve will display the current `GFlop/s` information in a pop-up window (as shown in the figure). If we only click on the curve, a vertical red line shows up, and we can read on it the `GFlop/s` values of all the curves at the same time.

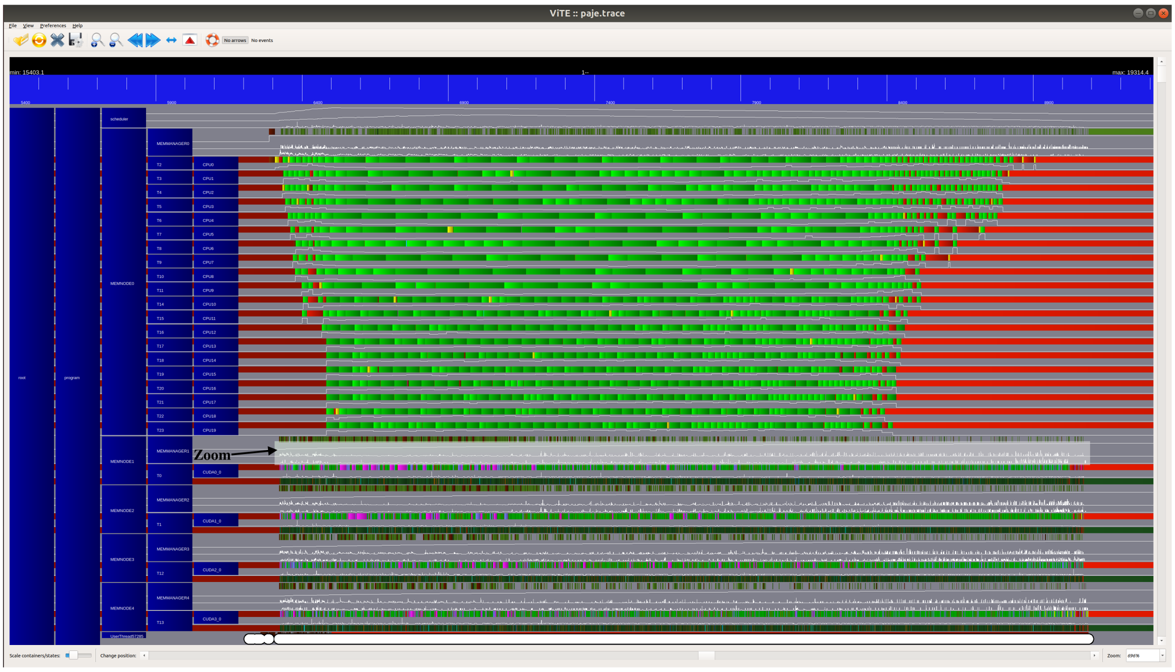


File opened: paje.trace

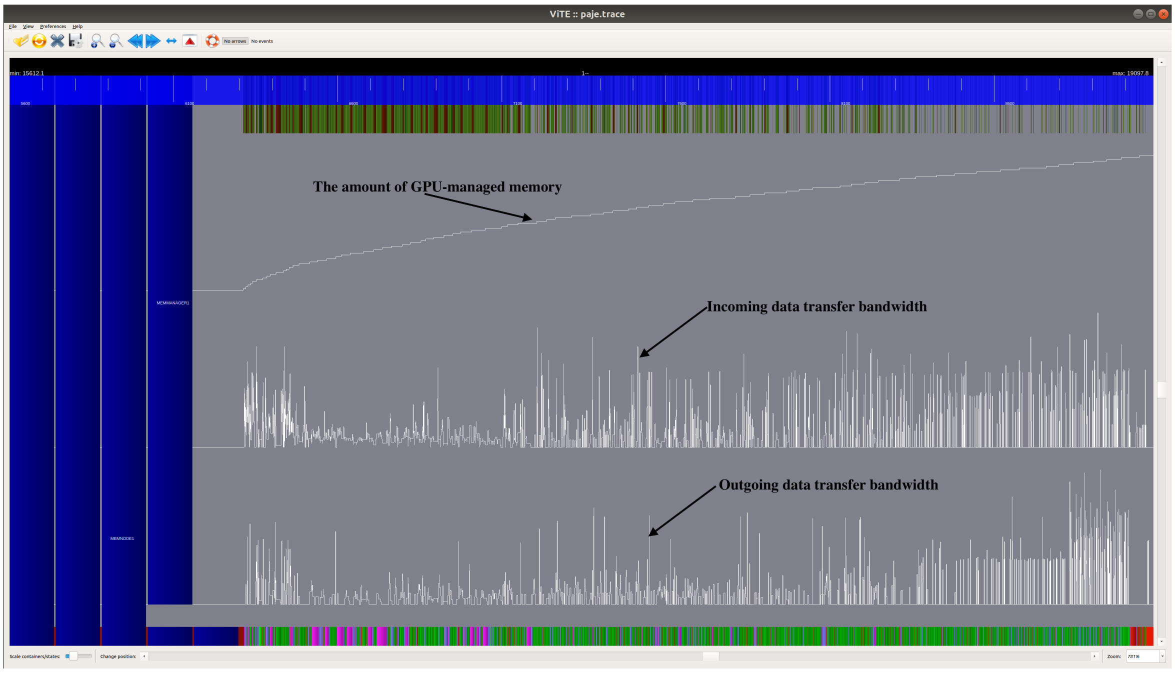
0 errors and 0 warnings were found during parsing.

Variable
Container: CUDA0_0
Type: GFlop/s
Value: 2246.27
Value: 2246.27
Min: 0
Max: 18595.5

For GPUs, there are three additional curves above the task blocks that can be double-clicked to open a pop-up window to view information. Let's zoom in on the three curves during the entire execution process as illustrated in the figure:



As shown in the figure below, the top curve represents the amount of GPU-managed memory in MBytes, while the bottom two curves represent the data transfer between tasks on the CPU and GPU, and between tasks on different GPUs. They respectively indicate the incoming and outgoing data transfer bandwidth. By looking at the memory curve, we can observe that the memory usage kept increasing at first, but due to the reutilization of the allocations by StarPU, the curve gradually became stable later on.

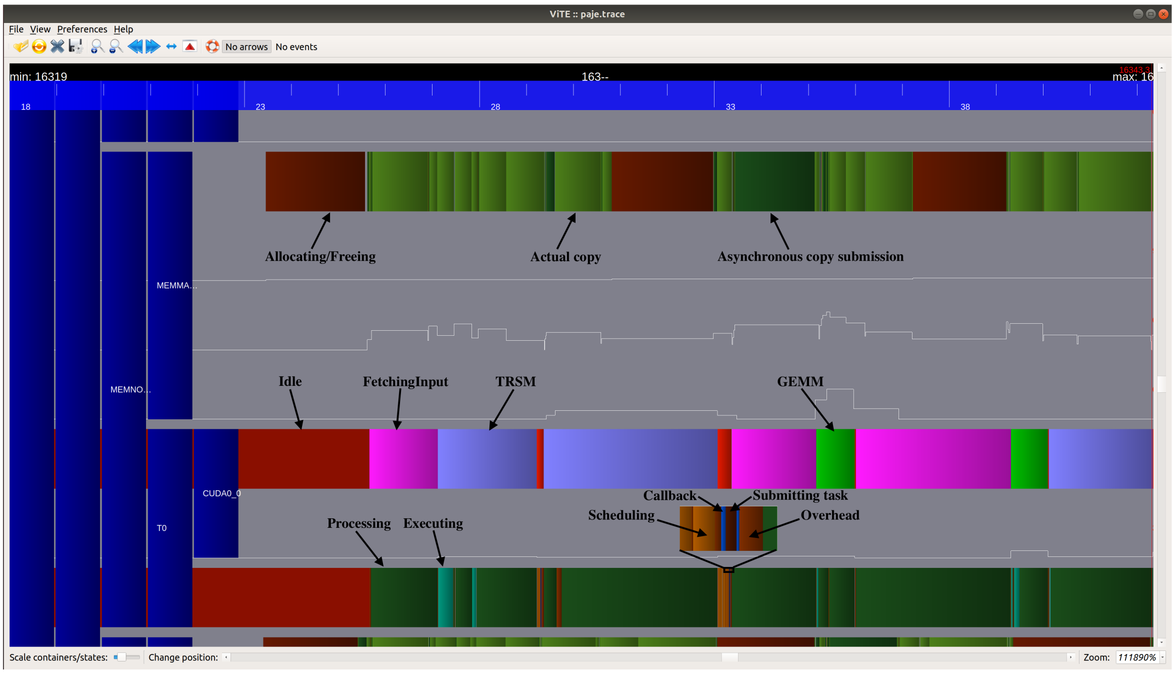

ViTE :: paje.trace
File View Preferences Help
The amount of GPU-managed memory
Incoming data transfer bandwidth
Outgoing data transfer bandwidth
MEMMANAGER1
MEMNODE1

23.1.1.4 States in Gantt Diagram

Above these three curves, we can see some blocks which represent driver copy (see the top of the figure below), i.e. a memory copy. The light green blocks represent the actual copies, the dark green blocks represent asynchronous copy submissions, and the burgundy blocks represent allocating and freeing. Double-clicking on a block allows us to view relevant information in the pop-up window.

Here, a couple of issues may show up:

- If the "Allocating/Freeing" parts take a long time, it means that StarPU does not manage to re-use data buffers allocated in the GPU. If you have e.g. a lot of tiles with different sizes, it may be useful to approximate the allocation size, by using e.g. starpu_matrix_data_register_allocsize() with the proper nx / ld / ny, but an allocation size that is rounded up, so that buffers with that same rounded size can be shared.
- If the "Asynchronous copy submission" parts take a long time, it means that the CPU buffers are not pinned: you need to make sure to use starpu_malloc(), or starpu_memory_pin() (see CUDA-specific Optimizations) so that the CPU buffers are pinned so that the GPU driver can efficiently process transfers asynchronously (in the "Actual copy" part) rather than synchronously (in the "Asynchronous copy submission" part).



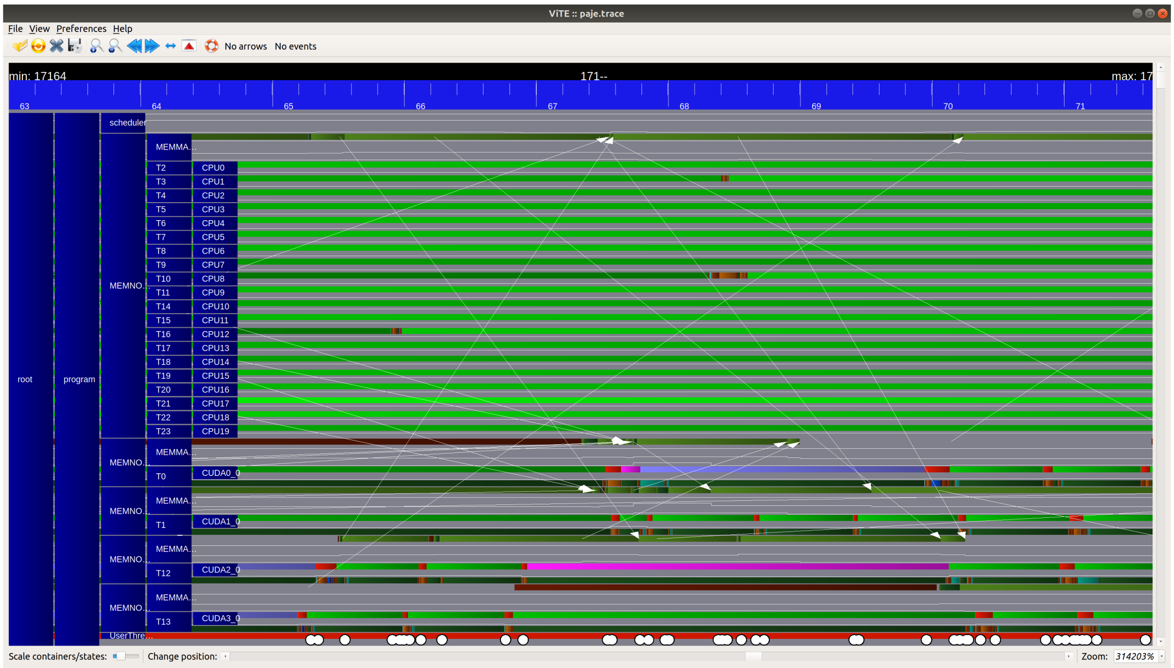
Below the GPU task blocks and `GFlops` curve (see the bottom of the figure above), we can see some other blocks that represent the CPU waiting for the GPU to complete the task. During time, CPU can do variable actions which are represented by blocks of different colors, such as:

- Dark green represents `progressing`, it keeps polling for task or data transfer completion
- Brown-yellow represents `scheduling`
- Burgundy represents `submitting task`
- Lake blue represents `executing`, it is executing the application codelet function. Here it is very short because the codelet just submits a kernel asynchronously.
- Dark blue represents `callback`
- Chestnut represents `overhead`. This state is not supposed to be long, as it represents everything that we did not classify as an operation that is supposed to be long like the operations mentioned above. If you find situations where some overhead is long, this is a bug worth reporting so we can fix it.

and we can always double-click on the block to view relevant information in the pop-up window.

23.1.1.5 Transfers in Gantt Diagram

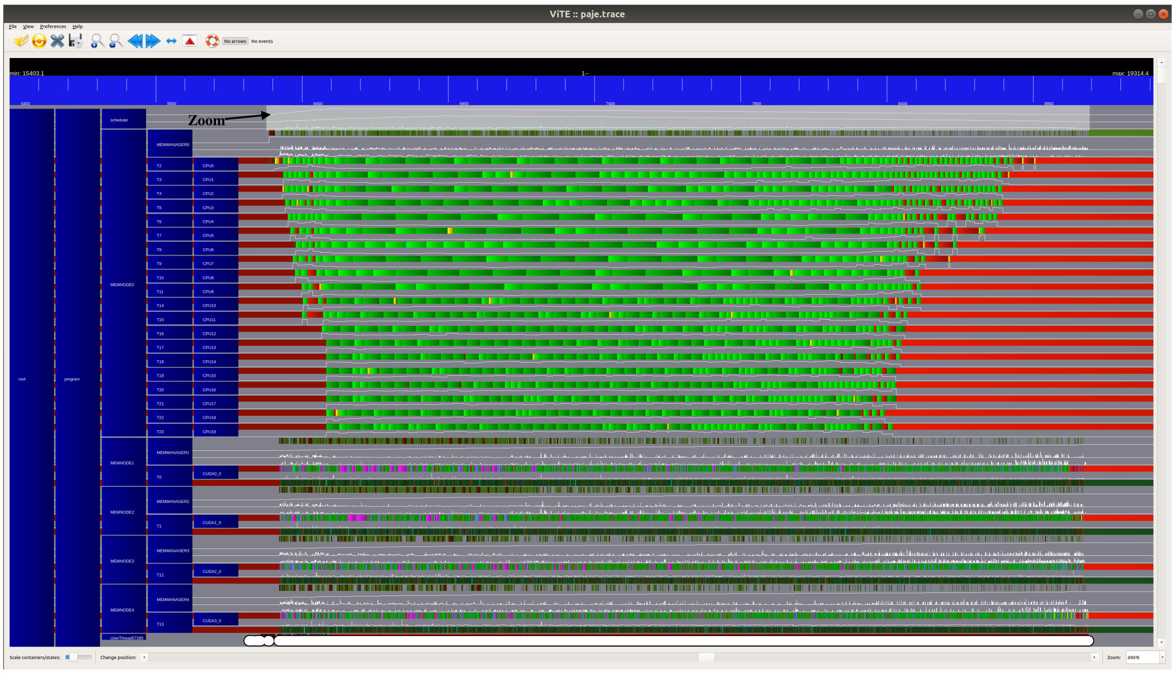
We can horizontally zoom in on a section of the Gantt diagram, and deselect the "No arrows" option. This will allow us to see a complete process of data transfer, as shown in the following figure:



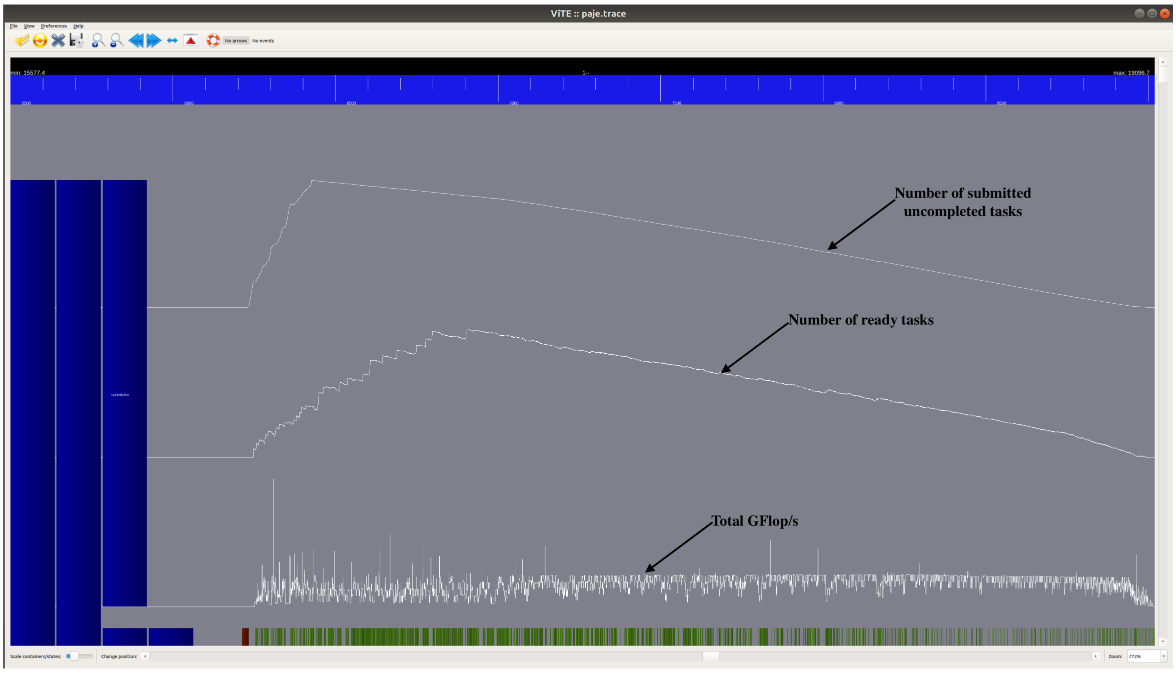
In the above figure, we can see a long segment of magenta color in CUDA2_0 task blocks. At the same time, we can see that there are numerous transfers between other workers during this time period. This indicates that CUDA2_0 is waiting for the completion of the data transfers needed by the task it wants to execute.

23.1.1.6 Scheduler in Gantt Diagram

At the top of the entire Gantt diagram, there are three curves that represent the information of the scheduler. Let's zoom in on the three curves during the entire execution process as illustrated in the figure below:

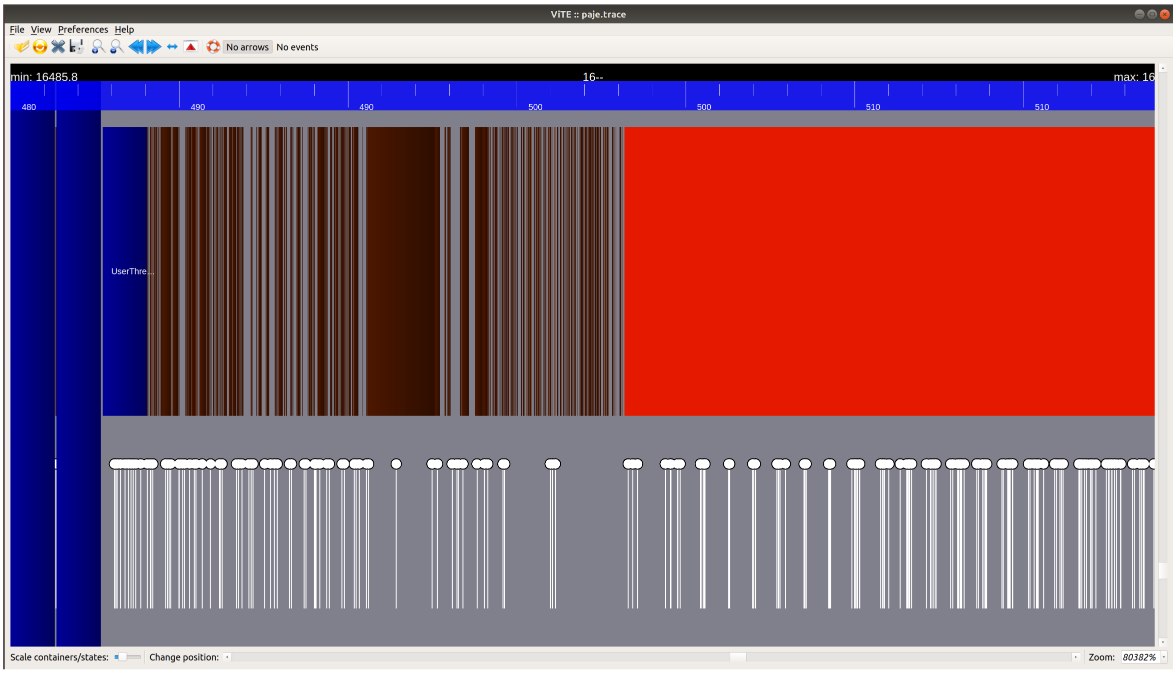


As shown in the figure below, from top to bottom, they respectively indicate the number of submitted uncompleted tasks, the number of ready tasks, and the total `GFlop/s` for this moment. By double-clicking on the curves, we can view relevant information in the pop-up window.



23.1.1.7 Main Thread in Gantt Diagram

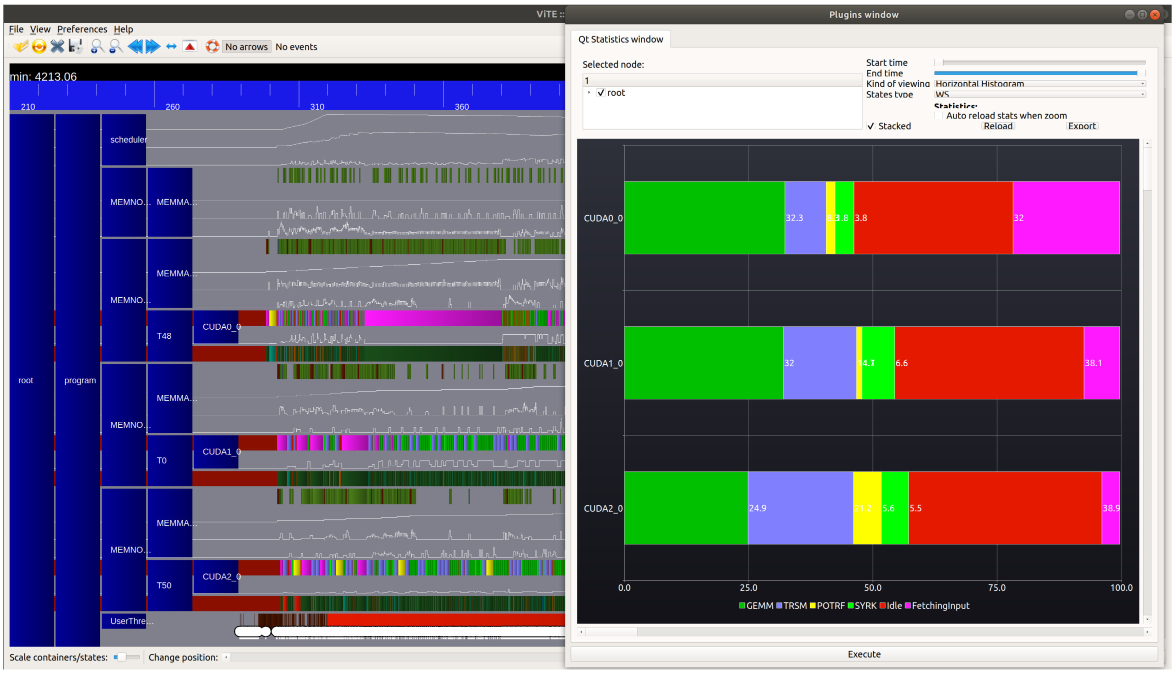
At the very bottom of the entire Gantt diagram, we will see a red bar, which represents the main thread waiting for tasks. In front of the red bar (see the figure below), there are some dark red bars, which represent the main thread submitting tasks.



Below these red bars, we can see some white vertical lines with small circles on top, which represent events. The default events can be either task push or task pop or task wait for all. The application can inject its own events at any desired moment with the function starpu_fxt_trace_user_event() or starpu_fxt_trace_user_event_string(). Similarly, double-clicking on the white bars allows you to see relevant information in the pop-up window.

23.1.1.8 Statistics in Gantt Diagram

To get statistics on the time spent in runtime overhead, we can use the statistics plugin of ViTE. In the Preferences menu, select Plugins. In "States Type", select "Worker State". Then click on "Reload" to update the histogram. The red "Idle" percentages are due to lack of parallelism, the "FetchingInput" percentages are due to waiting for data transfers. The brown "Overhead" and "Scheduling" percentages are due to the overhead of the runtime and of the scheduler.



23.1.2 Creating a DAG With Graphviz

Another generated trace file is a task graph described using the DOT language. The file, created in the current directory, is named `dag.dot` file in the current directory. It is possible to get a graphical output of the graph by using the `graphviz` library:

```
$ dot -Tpdf dag.dot -o output.pdf
```

23.1.3 Getting Task Details

Another generated trace file gives details on the executed tasks. The file, created in the current directory, is named `tasks.rec`. This file is in the `recutils` format, i.e. `Field:  value` lines, and empty lines are used to separate each task. This can be used as a convenient input for various ad-hoc analysis tools. By default, it only contains information about the actual execution. Performance models can be obtained by running `starpu_tasks_rec_complete` on it:

```
$ starpu_tasks_rec_complete tasks.rec tasks2.rec
```

which will add `EstimatedTime` lines which contain the performance model-estimated time (in µs) for each worker starting from 0. Since it needs the performance models, it needs to be run the same way as the application execution, or at least with `STARPU_HOSTNAME` set to the hostname of the machine used for execution, to get the performance models of that machine.
Another possibility is to obtain the performance models as an auxiliary `perfmodel.rec` file, by using the `starpu_perfmodel_recdump` utility:

```
$ starpu_perfmodel_recdump tasks.rec -o perfmodel.rec
```

One can also simply call starpu_task_get_name() to get the name of a task.

23.1.4 Getting Scheduling Task Details

The file, `sched_tasks.rec`, created in the current directory, in the `recutils` format, gives information about the tasks scheduling, and lists the push and pop actions of the scheduler. For each action, it gives the timestamp, the job priority and the job id. Each action is separated from the next one by empty lines. The job id associated with the task can be retrieved by calling starpu_task_get_job_id().

23.1.5 Monitoring Activity

Another generated trace file is an activity trace. The file, created in the current directory, is named `activity.data`. A profile of the application showing the activity of StarPU during the execution of the program can be generated:

```
$ starpu_workers_activity activity.data
```

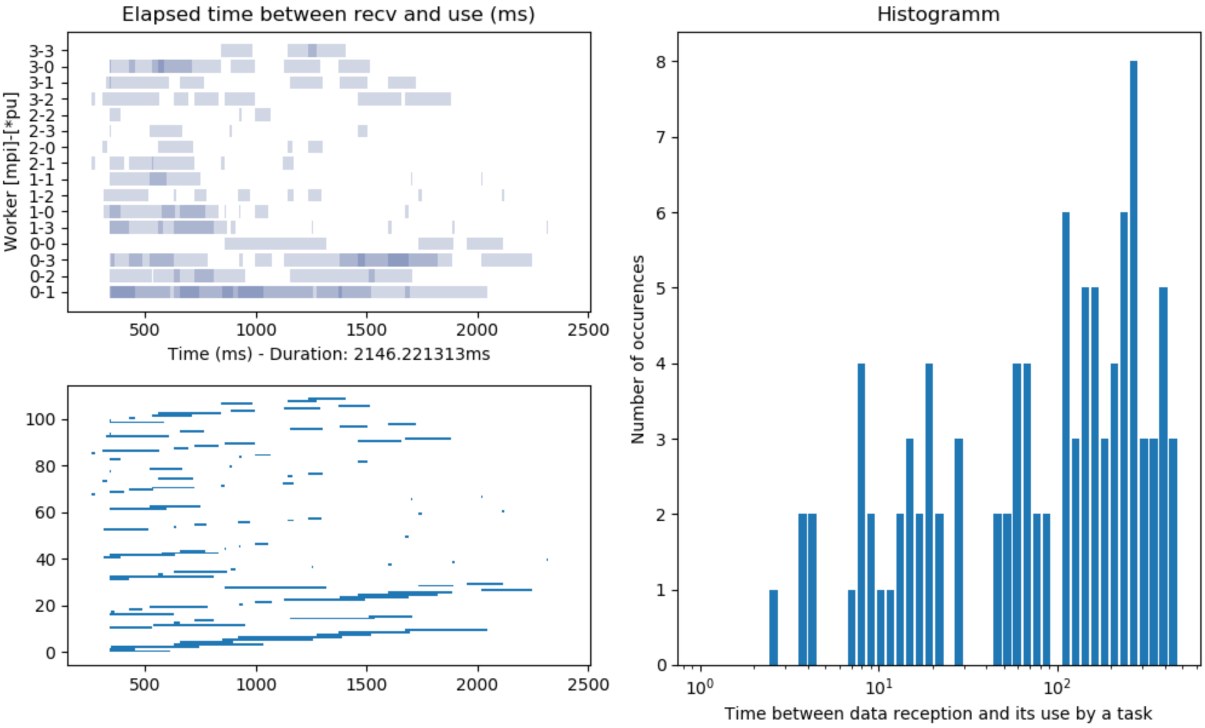
This will create a file named `activity.eps` in the current directory. This picture is composed of two parts. The first part shows the activity of the different workers. The green sections indicate which proportion of the time was spent executed kernels on the processing unit. The red sections indicate the proportion of time spent in StarPU: an important overhead may indicate that the granularity may be too low, and that bigger tasks may be appropriate to use the processing unit more efficiently. The black sections indicate that the processing unit was blocked because there was no task to process: this may indicate a lack of parallelism, which may be alleviated by creating more tasks when it is possible.
The second part of the picture `activity.eps` is a graph showing the evolution of the number of tasks available in the system during the execution. Ready tasks are shown in black, and tasks that are submitted but not schedulable yet are shown in grey.

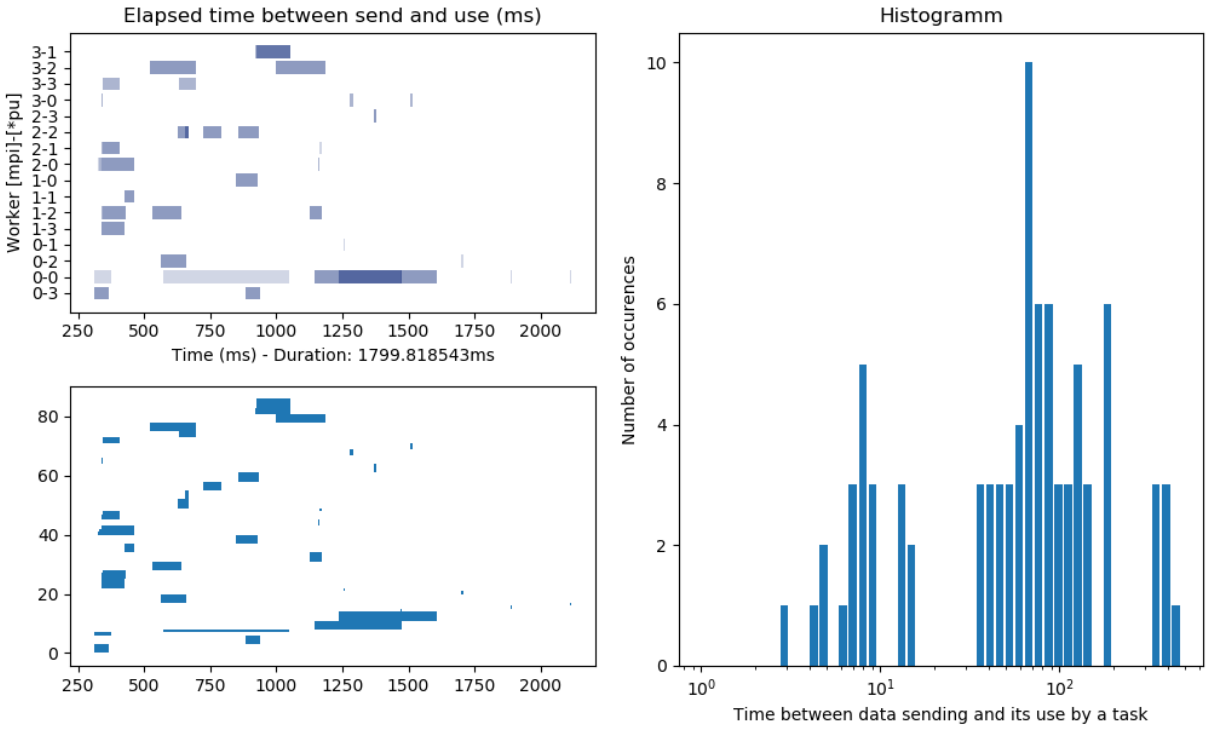
23.1.6 Getting Modular Schedular Animation

When using modular schedulers (i.e. schedulers which use a modular architecture, and whose name start with "modular-"), the call to `starpu_fxt_tool` will also produce a `trace.html` file which can be viewed in a javascript-enabled web browser. It shows the flow of tasks between the components of the modular scheduler.

23.1.7 Analyzing Time Between MPI Data Transfer and Use by Tasks

`starpu_fxt_tool` produces a file called `comms.rec` which describes all MPI communications. The script `starpu_send_recv_data_use.py` uses this file and `tasks.rec` in order to produce two graphs: the first one shows durations between the reception of data and their usage by a task and the second one plots the same graph but with elapsed time between send and usage of a data by the sender.





23.1.8 Number of events in trace files

When launched with the option `-number-events`, `starpu_fxt_tool` will produce a file named `number_events.data`. This file contains the number of events for each event type. Events are represented with their key. To convert event keys to event names, you can use the `starpu_fxt_number_events_to_names.py` script:

```
$ starpu_fxt_number_events_to_names.py number_events.data
```

The number of recorded events (and thus the performance overhead introduced by tracing) can be reduced by setting which categories of events to record with the environment variable STARPU_FXT_EVENTS.

23.1.9 Limiting The Scope Of The Trace

For computing statistics, it is useful to limit the trace to a given portion of the time of the whole execution. This can be achieved by calling

```
starpu_fxt_autostart_profiling(0)
```

before calling starpu_init(), to prevent tracing from starting immediately. Then

```
starpu_fxt_start_profiling();
```

and

```
starpu_fxt_stop_profiling();
```

can be used around the portion of code to be traced. This will show up as marks in the trace, and states of workers will only show up for that portion.

23.2 Performance Of Codelets

After calibrating performance models of codelets (see Performance Model Example and Performance Model Calibration), they can be examined by using the tool `starpu_perfmodel_display`:

```
$ starpu_perfmodel_display -l
file: <malloc_pinned.hannibal>
file: <starpu_slu_lu_model_trsm_ru.hannibal>
file: <starpu_slu_lu_model_getrf.hannibal>
file: <starpu_slu_lu_model_gemm.hannibal>
file: <starpu_slu_lu_model_trsm_ll.hannibal>
```

Here, the codelets of the example `lu` are available. We can examine the performance of the kernel `22` (in microseconds), which is history-based:

```
$ starpu_perfmodel_display -s starpu_slu_lu_model_gemm
performance model for cpu
# hash      size       mean          dev           n
57618ab0    19660800   2.851069e+05  1.829369e+04  109
performance model for cuda_0
# hash      size       mean          dev           n
57618ab0    19660800   1.164144e+04  1.556094e+01  315
performance model for cuda_1
# hash      size       mean          dev           n
57618ab0    19660800   1.164271e+04  1.330628e+01  360
performance model for cuda_2
# hash      size       mean          dev           n
57618ab0    19660800   1.166730e+04  3.390395e+02  456
```

We can see that for the given size, over a sample of a few hundreds of execution, the GPUs are about 20 times faster than the CPUs (numbers are in us). The standard deviation is extremely low for the GPUs, and less than 10% for CPUs.

This tool can also be used for regression-based performance models. It will then display the regression formula, and in the case of non-linear regression, the same performance log as for history-based performance models:

```
$ starpu_perfmodel_display -s non_linear_memset_regression_based
performance model for cpu_impl_0
        Regression : #sample = 1400
        Linear: y = alpha size ^ beta
                alpha = 1.335973e-03
                beta = 8.024020e-01
        Non-Linear: y = a size ^b + c
                a = 5.429195e-04
                b = 8.654899e-01
                c = 9.009313e-01
# hash          size            mean            stddev          n
a3d3725e        4096            4.763200e+00    7.650928e-01    100
870a30aa        8192            1.827970e+00    2.037181e-01    100
48e988e9        16384           2.652800e+00    1.876459e-01    100
961e65d2        32768           4.255530e+00    3.518025e-01    100
...
```

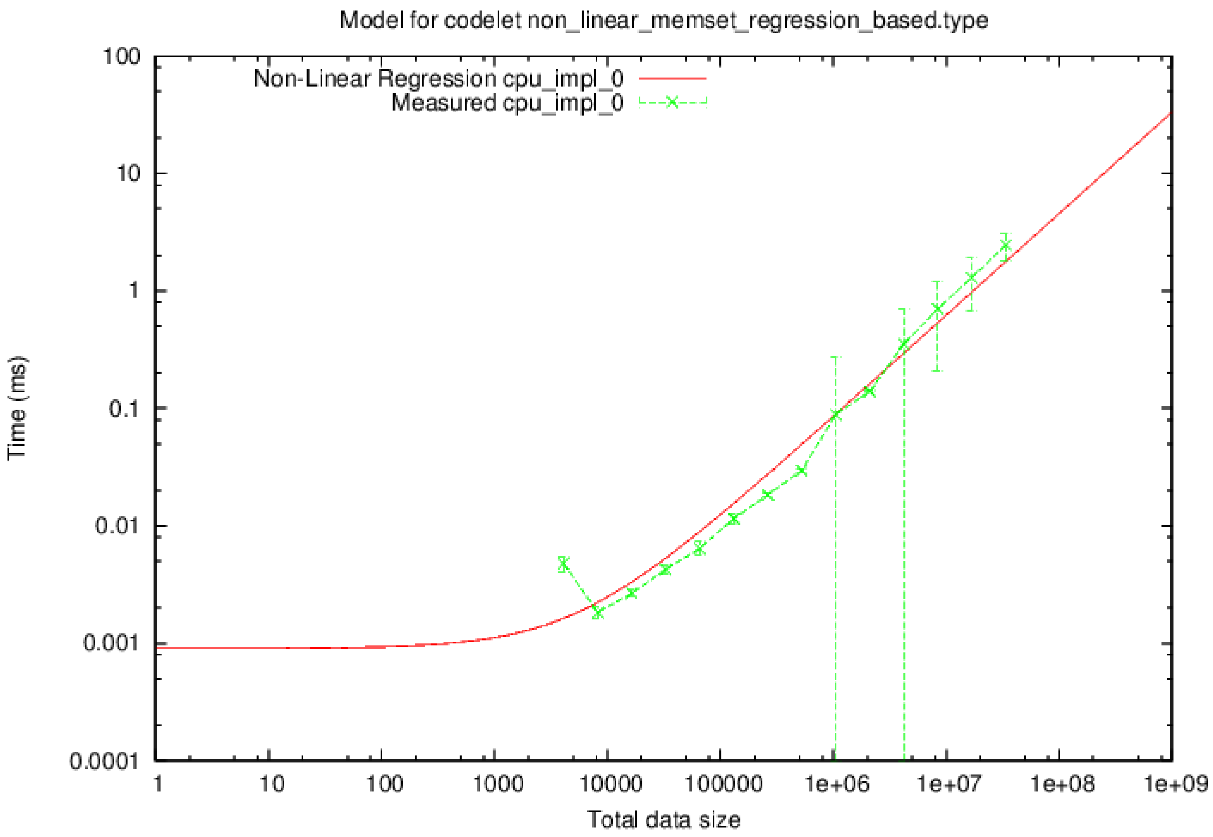
The same can also be achieved by using StarPU's library API, see Performance Model and notably the function starpu_perfmodel_load_symbol(). The source code of the tool `starpu_perfmodel_display` can be a useful example.

An XML output can also be printed by using the `-x` option:

```
$ tools/starpu_perfmodel_display -x -s non_linear_memset_regression_based
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE StarPUPerfmodel SYSTEM "starpu-perfmodel.dtd">
<!-- symbol non_linear_memset_regression_based -->
<!-- All times in us -->
<perfmodel version="45">
  <combination>
    <device type="CPU" id="0" ncores="1"/>
    <implementation id="0">
      <!-- cpu0_impl0 (Comb0) -->
      <!-- time = a size ^b + c -->
      <nl_regression a="5.429195e-04" b="8.654899e-01" c="9.009313e-01"/>
      <entry footprint="a3d3725e" size="4096" flops="0.000000e+00" mean="4.763200e+00" deviation="7.650928e-01
      <entry footprint="870a30aa" size="8192" flops="0.000000e+00" mean="1.827970e+00" deviation="2.037181e-01
      <entry footprint="48e988e9" size="16384" flops="0.000000e+00" mean="2.652800e+00" deviation="1.876459e-0
      <entry footprint="961e65d2" size="32768" flops="0.000000e+00" mean="4.255530e+00" deviation="3.518025e-0
    </implementation>
  </combination>
</perfmodel>
```

The tool `starpu_perfmodel_plot` can be used to draw performance models. It writes a `.gp` file in the current directory, to be run with the tool `gnuplot`, which shows the corresponding curve.

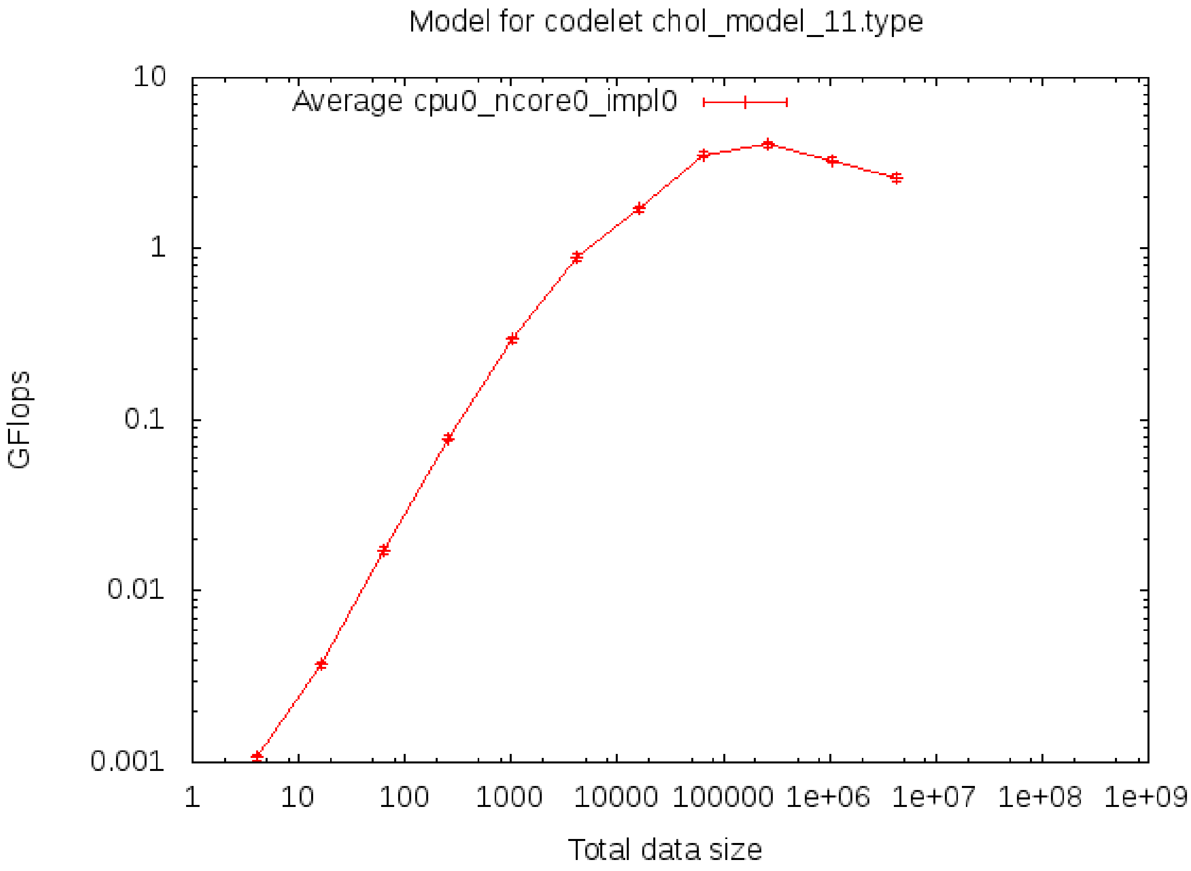
```
$ tools/starpu_perfmodel_plot -s non_linear_memset_regression_based
$ gnuplot starpu_non_linear_memset_regression_based.gp
$ gv starpu_non_linear_memset_regression_based.png
```



When the field starpu_task::flops is set (or STARPU_FLOPS is passed to starpu_task_insert()), `starpu_perfmodel_plot` can directly draw a GFlops/s curve, by simply adding the `-f` option:

```
$ starpu_perfmodel_plot -f -s chol_model_potrf
```

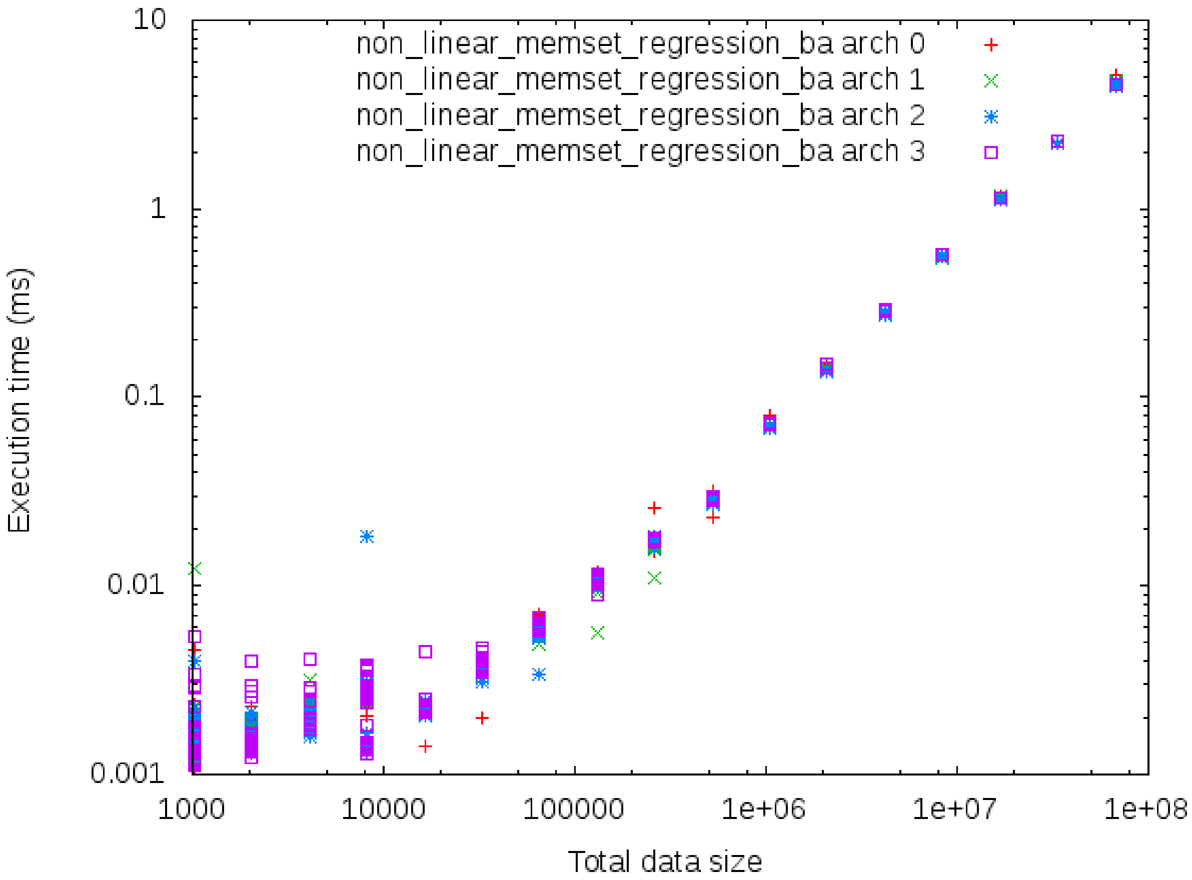
This will however disable displaying the regression model, for which we can not compute GFlops/s.



When the FxT trace file `prof_file_something` has been generated, it is possible to get a profiling of each codelet by calling:

```
$ starpu_fxt_tool -i /tmp/prof_file_something
$ starpu_codelet_profile distrib.data codelet_name
```

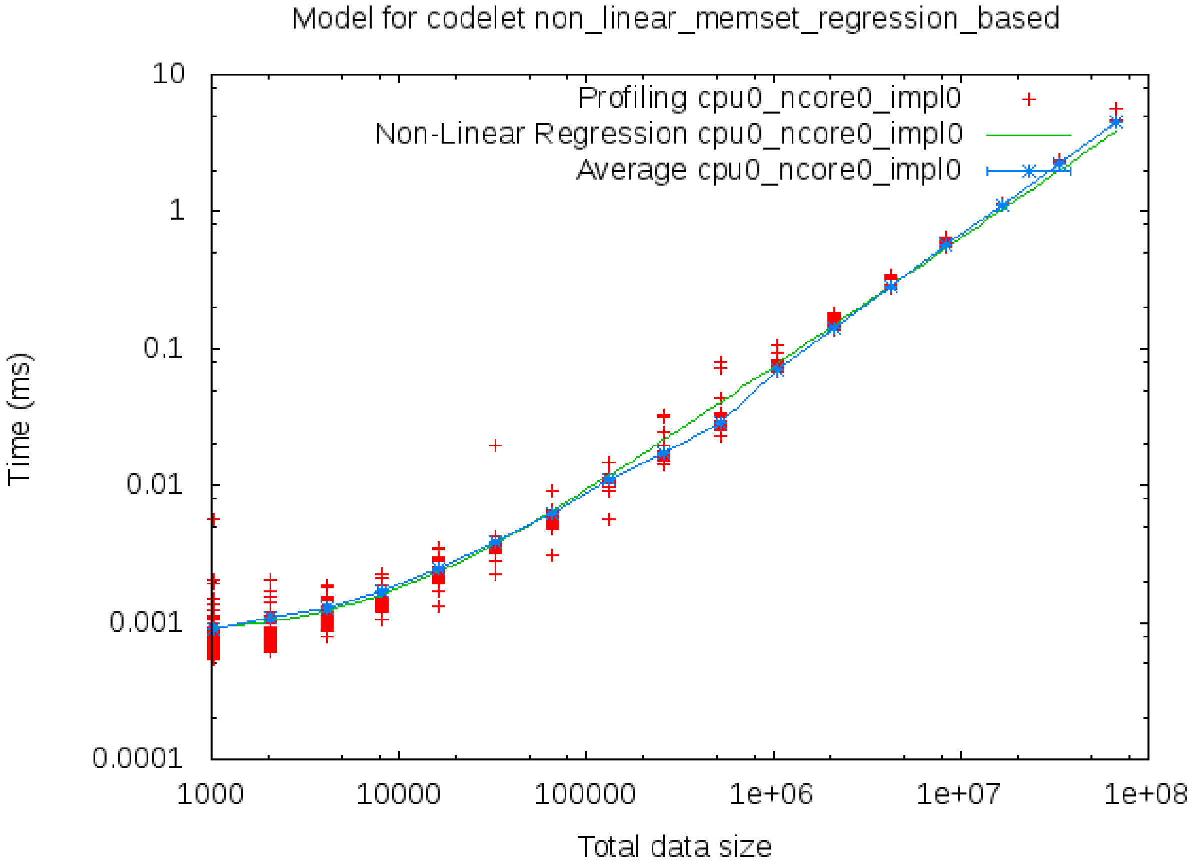
This will create profiling data files, and a `distrib.data.gp` file in the current directory, which draws the distribution of codelet time over the application execution, according to data input size.



This is also available in the tool `starpu_perfmodel_plot`, by passing it the fxt trace:

```
$ starpu_perfmodel_plot -s non_linear_memset_regression_based -i /tmp/prof_file_foo_0
```

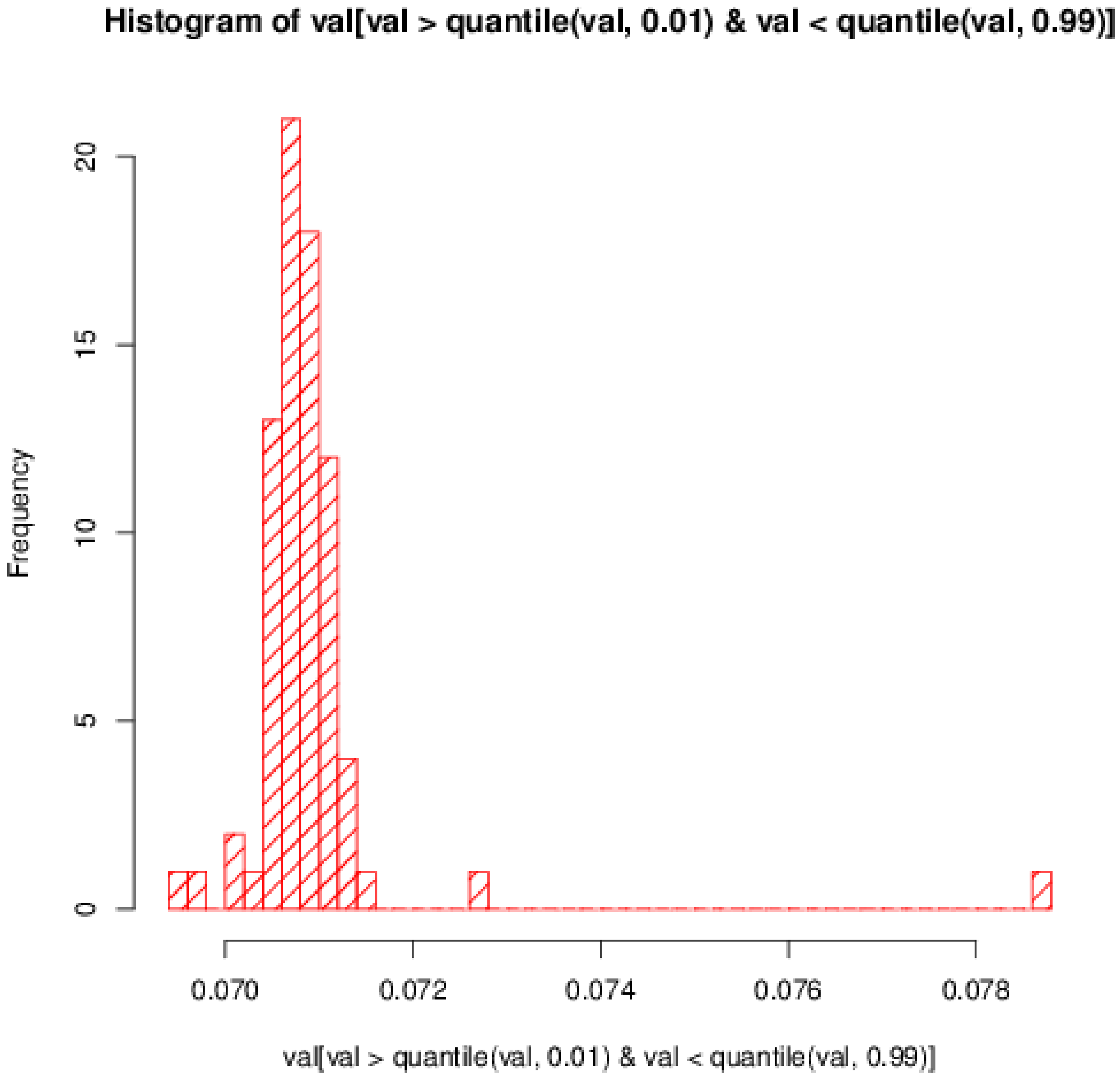
It will produce a `.gp` file which contains both the performance model curves, and the profiling measurements.



If you have the statistical tool `R` installed, you can additionally use

```
$ starpu_codelet_histo_profile distrib.data
```

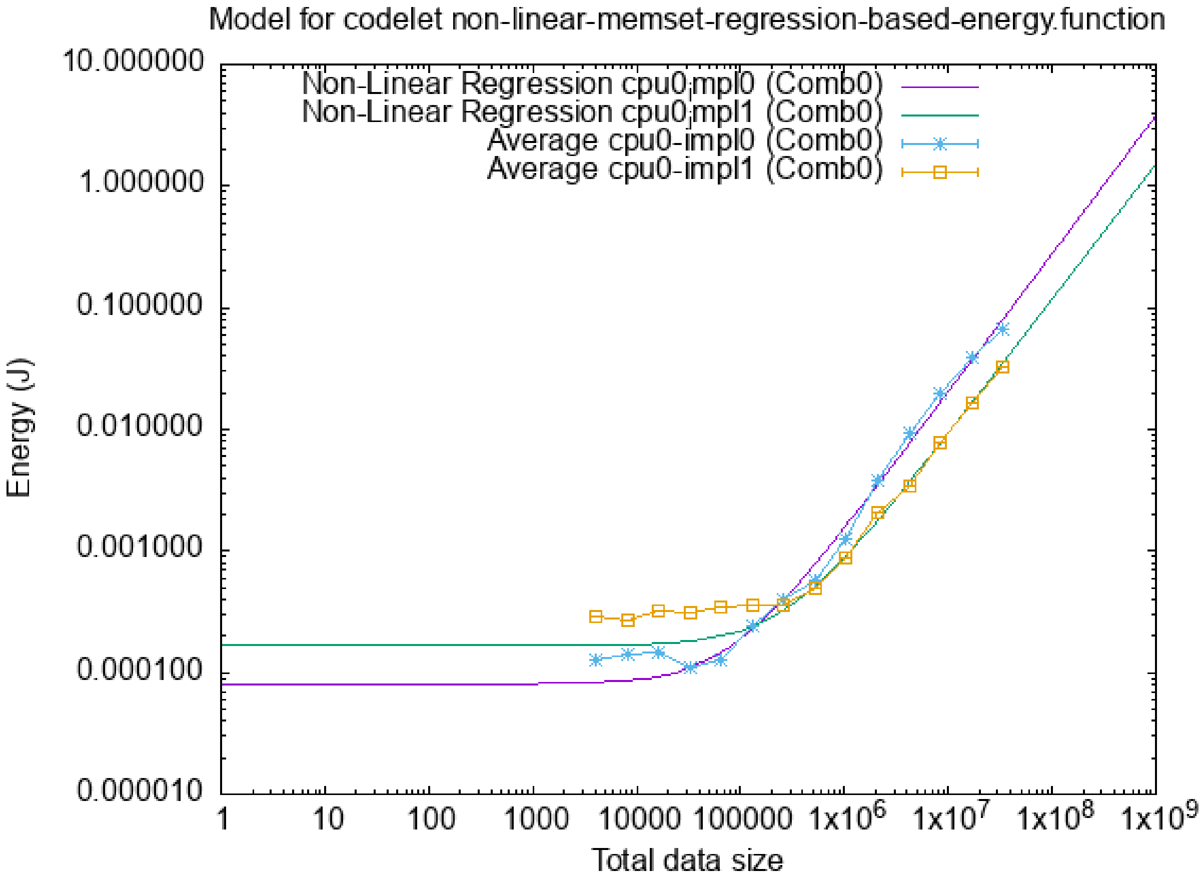
Which will create one `.pdf` file per codelet and per input size, showing a histogram of the codelet execution time distribution.



23.3 Energy Of Codelets

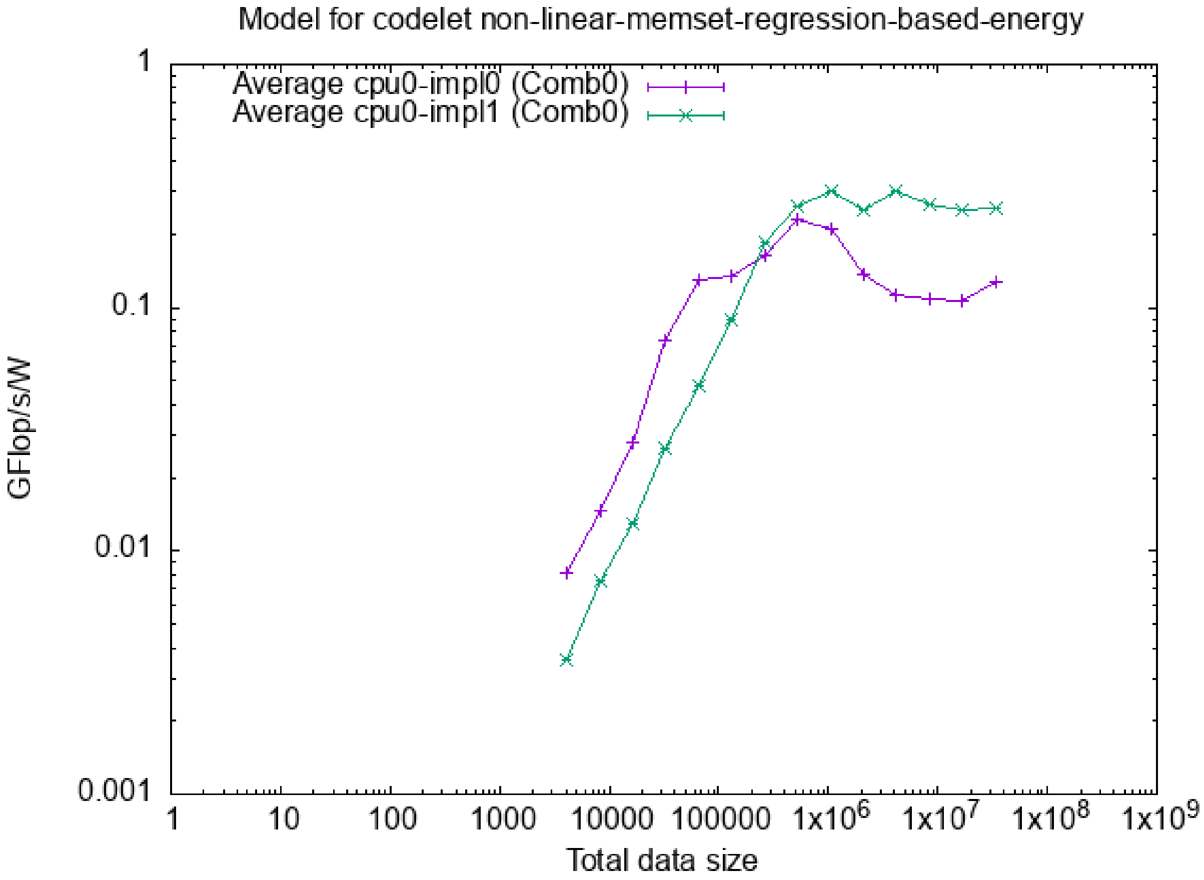
A performance model of the energy of codelets can also be recorded thanks to the starpu_codelet::energy_model field of the starpu_codelet structure. StarPU usually cannot record this automatically, since the energy measurement probes are usually not fine-grain enough. It is however possible to measure it by writing a program that submits batches of tasks, let StarPU measure the energy requirement of the batch, and compute an average, see Measuring energy and power with StarPU .
The energy performance model can then be displayed in Joules with `starpu_perfmodel_display` just like the time performance model. The `starpu_perfmodel_plot` needs an extra `-e` option to display the proper unit in the graph:

```
$ tools/starpu_perfmodel_plot -e -s non_linear_memset_regression_based_energy
$ gnuplot starpu_non_linear_memset_regression_based_energy.gp
$ gv starpu_non_linear_memset_regression_based_energy.png
```



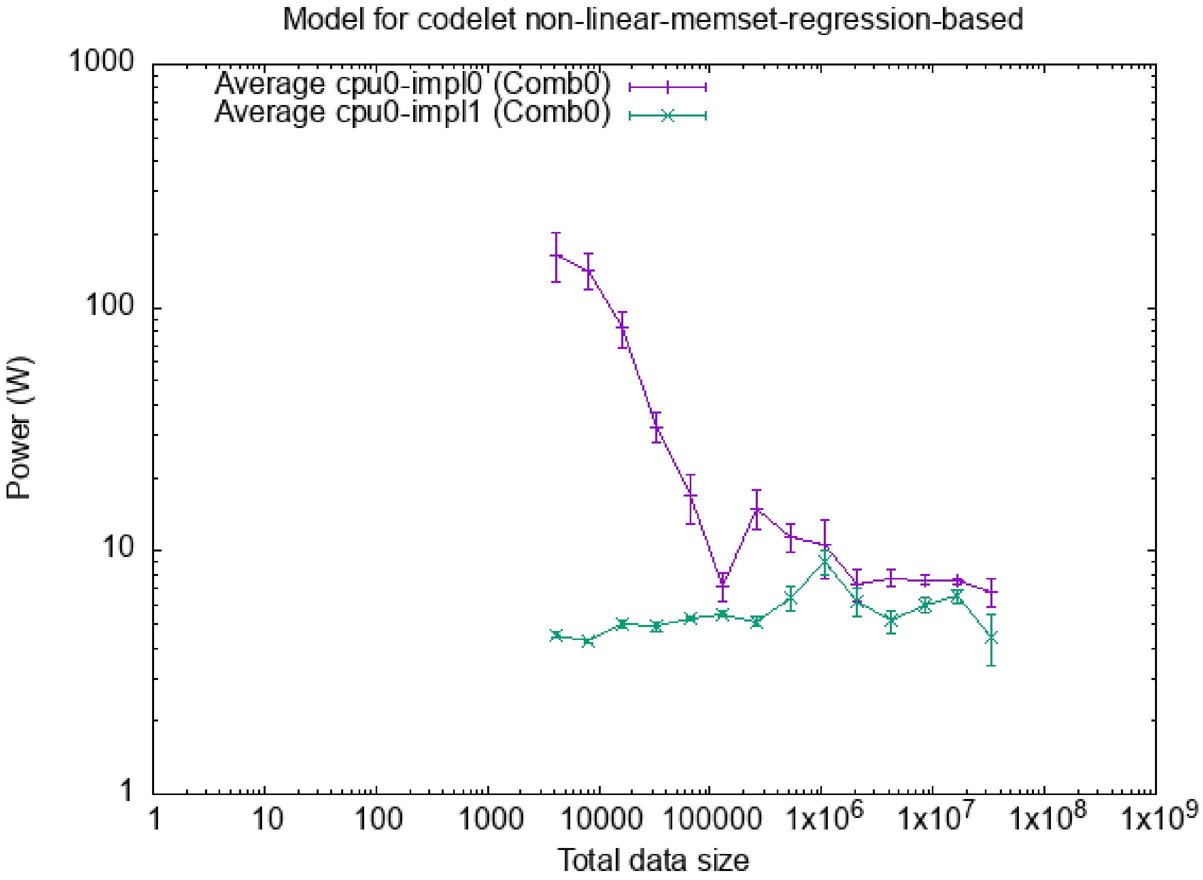
The `-f` option can also be used to display the performance in terms of GFlops/s/W, i.e. the efficiency:

```
$ tools/starpu_perfmodel_plot -f -e -s non_linear_memset_regression_based_energy
$ gnuplot starpu_gflops_non_linear_memset_regression_based_energy.gp
$ gv starpu_gflops_non_linear_memset_regression_based_energy.png
```



We clearly see here that it is much more energy-efficient to stay in the L3 cache.
One can combine the two time and energy performance models to draw Watts:

```
$ tools/starpu_perfmodel_plot -se non_linear_memset_regression_based non_linear_memset_regression_based_energy
$ gnuplot starpu_power_non_linear_memset_regression_based.gp
$ gv starpu_power_non_linear_memset_regression_based.eps
```

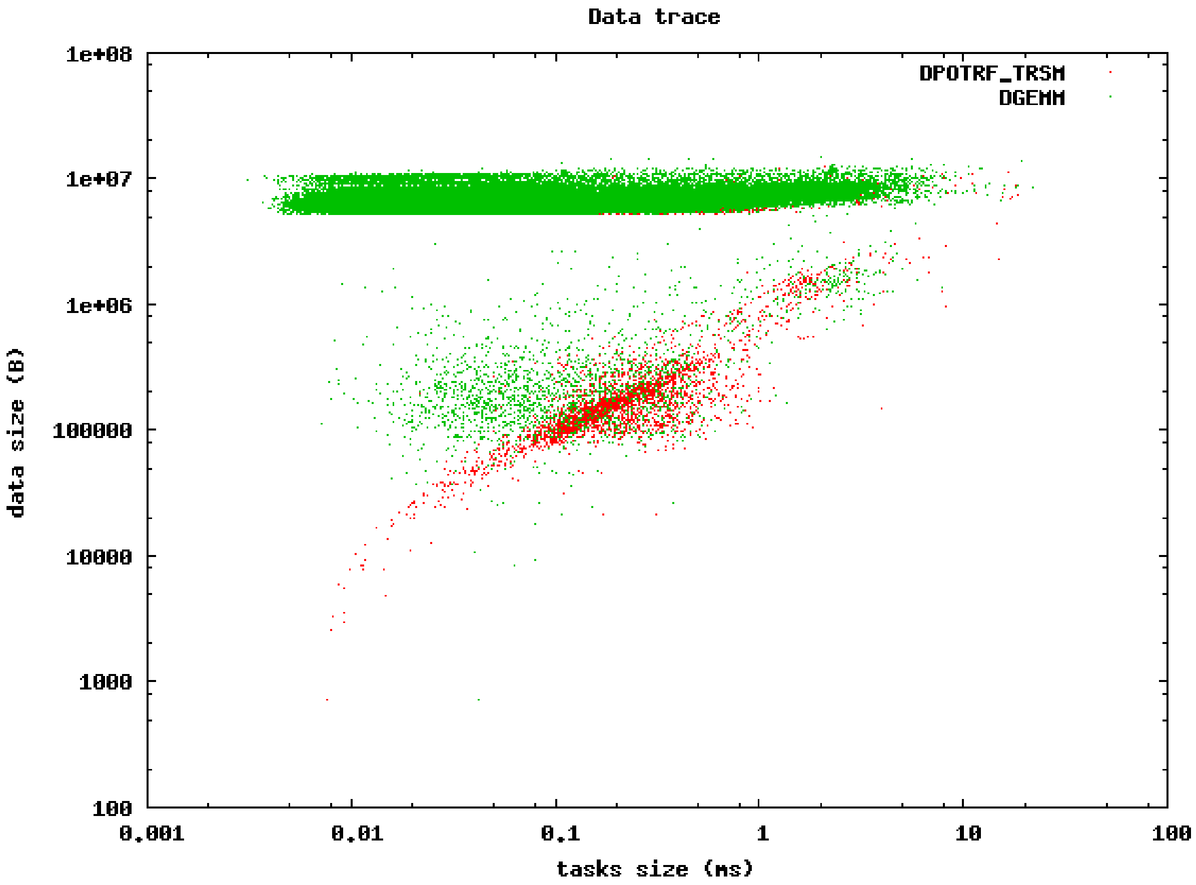


23.4 Data trace and tasks length

It is possible to get statistics about tasks length and data size by using :

```
$ starpu_fxt_data_trace filename [codelet1 codelet2 ... codeletn]
```

Where filename is the FxT trace file and codeletX the names of the codelets you want to profile (if no names are specified, `starpu_fxt_data_trace` will profile them all). This will create a file, `data_trace.gp` which can be executed to get a `.eps` image of these results. On the image, each point represents a task, and each color corresponds to a codelet.



23.5 Trace Statistics

More than just codelet performance, it is interesting to get statistics over all kinds of StarPU states (allocations, data transfers, etc.). This is particularly useful to check what may have gone wrong in the accuracy of the SimGrid simulation.
This requires the `R` statistical tool, with the `plyr`, `ggplot2` and `data.table` packages. If your system distribution does not have packages for these, one can fetch them from `CRAN`:

```
$ R
> install.packages("plyr")
> install.packages("ggplot2")
> install.packages("data.table")
> install.packages("knitr")
```

The `pj_dump` tool from `pajeng` is also needed (see `https://github.com/schnorr/pajeng`)
One can then get textual or `.csv` statistics over the trace states:

```
$ starpu_paje_state_stats -v native.trace simgrid.trace
"Value"            "Events_native.csv" "Duration_native.csv" "Events_simgrid.csv" "Duration_simgrid.csv"
"Callback"         220                 0.075978              220                  0
"chol_model_potrf" 10                  565.176               10                   572.8695
"chol_model_trsm"  45                  9184.828              45                   9170.719
"chol_model_gemm"  165                 64712.07              165                  64299.203
$ starpu_paje_state_stats native.trace simgrid.trace
```

An other way to get statistics of StarPU states (without installing `R` and `pj_dump`) is to use the `starpu_trace_state_stats.py` script, which parses the generated `trace.rec` file instead of the `paje.trace` file. The output is similar to the previous script, but it doesn't need any dependencies.
The different prefixes used in `trace.rec` are:

```
E: Event type
N: Event name
```

```
C: Event category
W: Worker ID
T: Thread ID
S: Start time
```

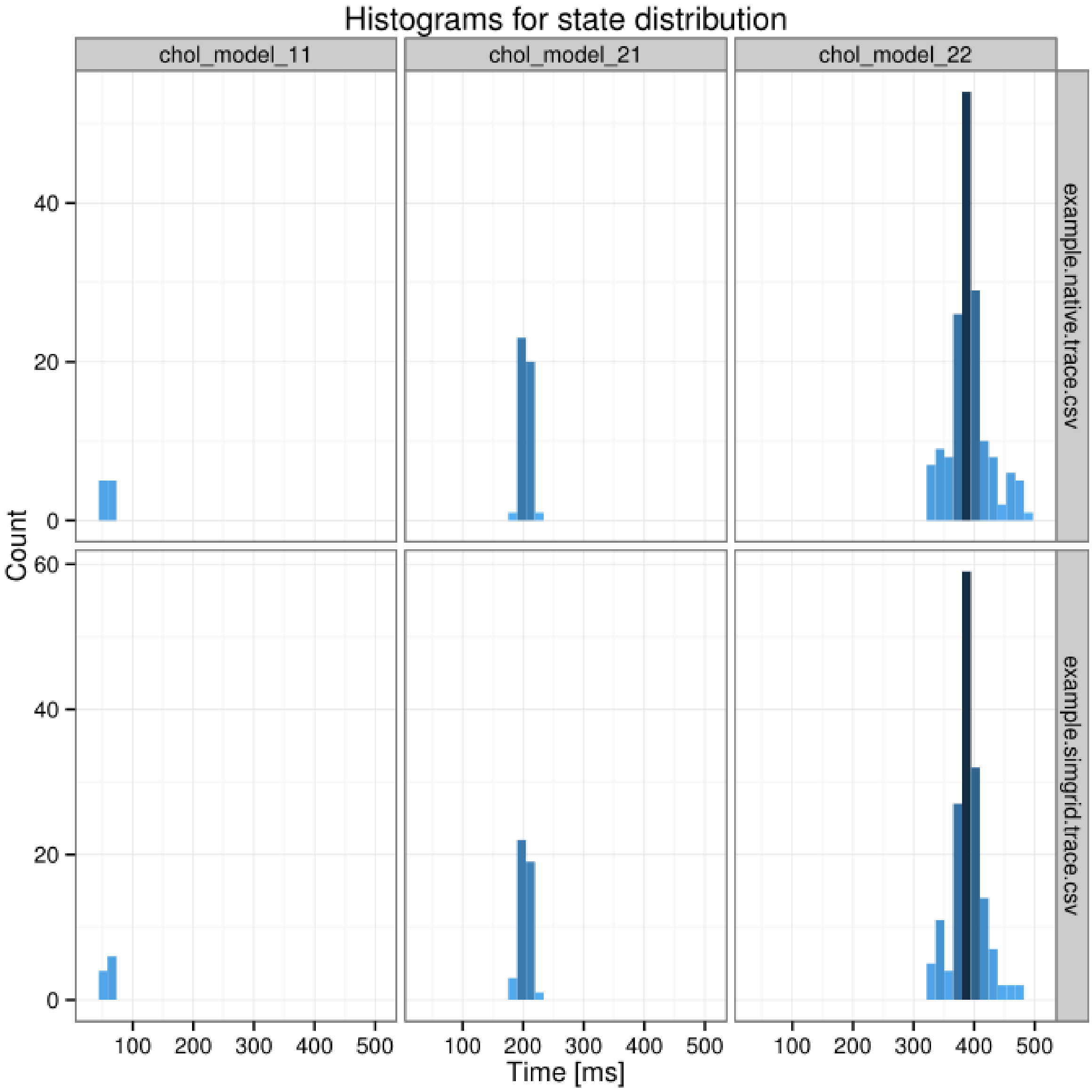
Here's an example on how to use it:

```
$ starpu_trace_state_stats.py trace.rec | column -t -s ","
"Name"              "Count" "Type"       "Duration"
"Callback"           220 Runtime 0.075978
"chol_model_potrf"   10  Task     565.176
"chol_model_trsm"    45  Task     9184.828
"chol_model_gemm"    165 Task     64712.07
```

`starpu_trace_state_stats.py` can also be used to compute the different efficiencies. Refer to the usage description to show some examples.
And one can plot histograms of execution times, of several states, for instance:

```
$ starpu_paje_draw_histogram -n chol_model_potrf,chol_model_trsm,chol_model_gemm native.trace simgrid.trace
```

and see the resulting pdf file:



A quick statistical report can be generated by using:

```
$ starpu_paje_summary native.trace simgrid.trace
```

it includes gantt charts, execution summaries, as well as state duration charts and time distribution histograms.
Other external Paje analysis tools can be used on these traces, one just needs to sort the traces by timestamp order (which not guaranteed to make recording more efficient):

```
$ starpu_paje_sort paje.trace
```

23.6 PAPI counters

Performance counter values could be obtained from the PAPI framework if `./configure` detected the libpapi.
In Debian, the `libpapi-dev` package provides the required files. Additionally, the `papi-tools` package contains a set of useful tools, for example `papi_avail` to see which counters are available.
To be able to use Papi counters, one may need to reduce the level of the kernel parameter `kernel.perf_event_paranoid` to 2 or below. See `https://www.kernel.org/doc/html/latest/admin-guide/perf-secu`
`html` for the security impact of this parameter.
Then one has to set the STARPU_PROFILING environment variable to 1 and specify which events to record with the STARPU_PROF_PAPI_EVENTS environment variable. For instance:

```
export STARPU_PROFILING=1 STARPU_PROF_PAPI_EVENTS="PAPI_TOT_INS PAPI_TOT_CYC"
```

The comma can also be used to separate events to monitor.
In the current simple implementation, only CPU tasks have their events measured and require CPUs that support the PAPI events. It is important to note that not all events are available on all systems, and general PAPI recommendations should be followed.
The counter values can be accessed using the profiling interface:
`task->profiling_info->papi_values`
Also, it can be accessed and/or saved with tracing when using STARPU_FXT_TRACE. With the use of `starpu_fxt_tool` the file `papi.rec` is generated containing the following triple:

```
Task Id
Event Id
Value
```

External tools like `rec2csv` can be used to convert this rec file to a `csv` file, where each line represents a value for an event for a task.

23.7 Theoretical Lower Bound On Execution Time

StarPU can record a trace of what tasks are needed to complete the application, and then, by using a linear system, provide a theoretical lower bound of the execution time (i.e. with an ideal scheduling).
The computed bound is not really correct when not taking into account dependencies, but for an application which have enough parallelism, it is very near to the bound computed with dependencies enabled (which takes a huge lot more time to compute), and thus provides a good-enough estimation of the ideal execution time.
Then there is an example to show how to use this.
For kernels with history-based performance models (and provided that they are completely calibrated), StarPU can very easily provide a theoretical lower bound for the execution time of a whole set of tasks. See for instance `examples/lu/lu_example.c`: before submitting tasks, call the function starpu_bound_start(), and after complete execution, call starpu_bound_stop(). starpu_bound_print_lp() or starpu_bound_print_mps() can then be used to output a Linear Programming problem corresponding to the schedule of your tasks. Or starpu_bound_print_dot() can be used to print a task dependency graph in the DOT format. Run it through `lp_solve` or any other linear programming solver, and that will give you a lower bound for the total execution time of your tasks. If StarPU was compiled with the library `glpk` installed, starpu_bound_compute() can be used to solve it immediately and get the optimized minimum, in ms. Its parameter `integer` allows deciding whether integer resolution should be computed and returned. Besides to solve it immediately and get the optimized minimum starpu_bound_print() can also print the statistics of actual execution and theoretical upper bound.
The `deps` parameter tells StarPU whether to take tasks, implicit data, and tag dependencies into account. Tags released in a callback or similar are not taken into account, only tags associated with a task are. It must be understood that the linear programming problem size is quadratic with the number of tasks and thus the time to solve it will be very long, it could be minutes for just a few dozen tasks. You should probably use `lp_solve`

`-timeout 1 test.pl -wmps test.mps` to convert the problem to MPS format and then use a better solver, `glpsol` might be better than `lp_solve` for instance (the `-pcost` option may be useful), but sometimes doesn't manage to converge. `cbc` might look slower, but it is parallel. For `lp_solve`, be sure to try at least all the `-B` options. For instance, we often just use `lp_solve -cc -B1 -Bb -Bg -Bp -Bf -Br -BG -Bd -Bs -BB -Bo -Bc -Bi` , and the `-gr` option can also be quite useful. The resulting schedule can be observed by using the tool `starpu_lp2paje`, which converts it into the Paje format.

Data transfer time can only be taken into account when `deps` is set. Only data transfers inferred from implicit data dependencies between tasks are taken into account. Other data transfers are assumed to be completely overlapped.

Setting `deps` to 0 will only take into account the actual computations on processing units. However, it still properly takes into account the varying performances of kernels and processing units, which is quite more accurate than just comparing StarPU performances with the fastest of the kernels being used.

The `prio` parameter tells StarPU whether to simulate taking into account the priorities as the StarPU scheduler would, i.e. schedule prioritized tasks before less prioritized tasks, to check to which extend this results to a less optimal solution. This increases even more computation time.

23.8 Trace visualization with StarVZ

Creating views with StarVZ (see: `https://github.com/schnorr/starvz`) is made up of two steps. The initial stage consists of a pre-processing of the traces generated by the application, while the second one consists of the analysis itself and is carried out with R packages' aid. StarVZ is available at CRAN (`https://cran.r-project.org/package=starvz`) and depends on `pj_dump` (from `pajeng`) and `rec2csv` (from `recutils`).

To download and install StarVZ, it is necessary to have `R`, `pajeng`, and `recutils`:

```
# For pj_dump and rec2csv
apt install -y pajeng recutils

# For R
apt install -y r-base libxml2-dev libssl-dev libcurl4-openssl-dev libgit2-dev libboost-dev
```

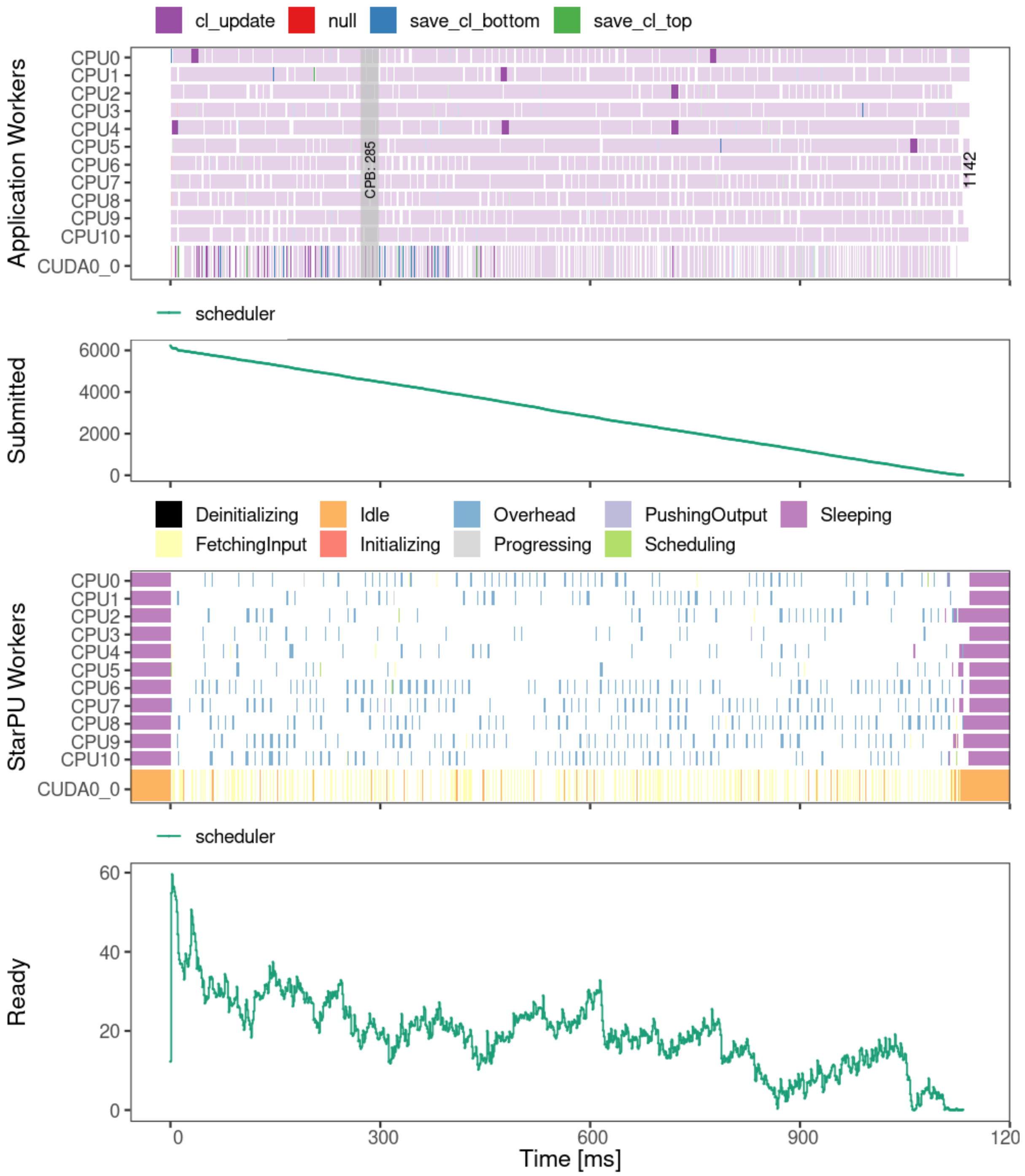
To install the StarVZ, the following command can be used:

```
echo "install.packages('starvz', repos = 'https://cloud.r-project.org')" | R --vanilla
```

To generate traces from an application, it is necessary to set STARPU_GENERATE_TRACE and build StarPU with FxT. Then, StarVZ can be used on a folder with StarPU FxT traces to produce a default view:

```
export PATH=$(Rscript -e 'cat(system.file("tools/", package = "starvz"), sep="\n")'):$PATH

starvz /foo/path-to-fxt-files
```

An example of default view:



One can also use existing trace files (`paje.trace`, `tasks.rec`, `data.rec`, `papi.rec` and `dag.dot`) skipping the StarVZ internal call to starpu_fxt_tool with:

```
starvz --use-paje-trace /foo/path-to-trace-files
```

Alternatively, each StarVZ step can be executed separately. Step 1 can be used on a folder with:

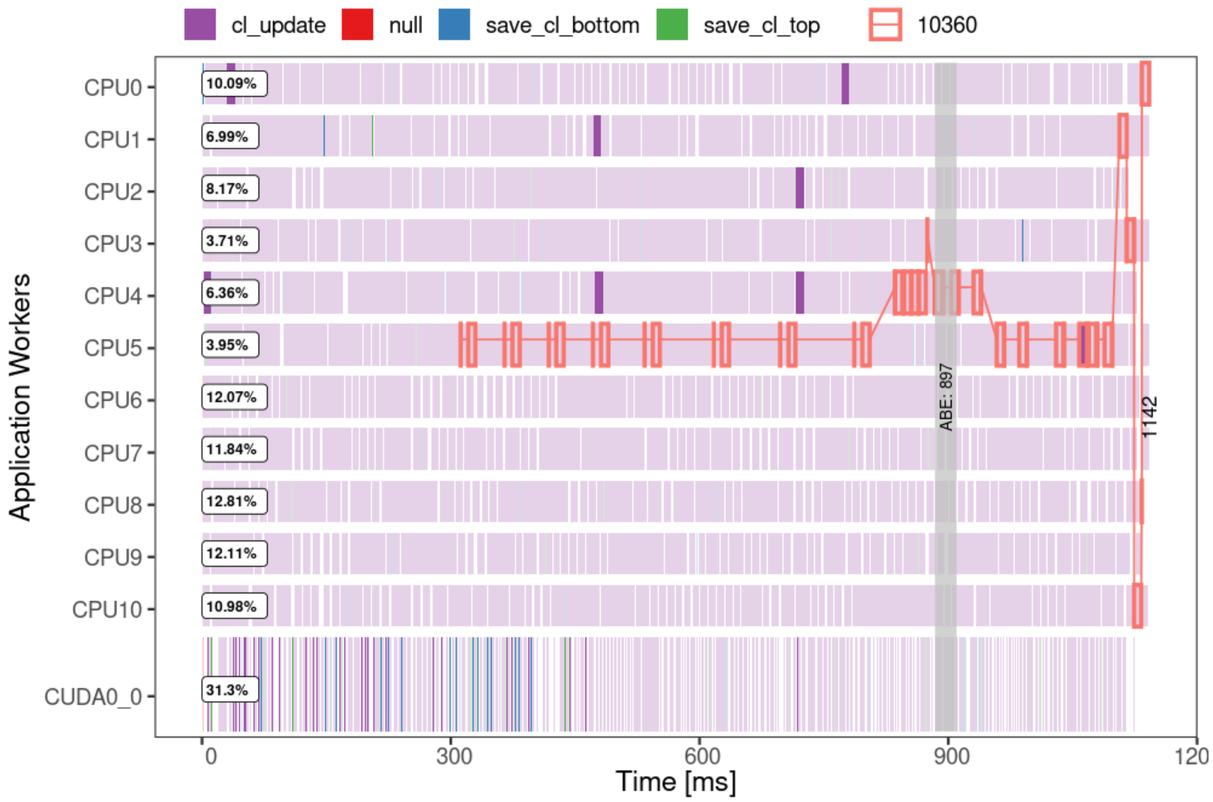
```
starvz -1 /foo/path-to-fxt-files
```

Then the second step can be executed directly in R. StarVZ enables a set of different plots that can be configured on a .yaml file. A default file is provided (`default.yaml`); also, the options can be changed directly in R.

```
library(starvz)
library(dplyr)
```

```
dtrace <- starvz_read("./", selective = FALSE)

# show idleness ratio
dtrace$config$st$idleness = TRUE

# show ABE bound
dtrace$config$st$abe$active = TRUE

# find the last task with dplyr
dtrace$config$st$tasks$list = dtrace$Application %>% filter(End == max(End)) %>% .$JobId
# show last task dependencies
dtrace$config$st$tasks$active = TRUE
dtrace$config$st$tasks$levels = 50

plot <- starvz_plot(dtrace)
```

An example of visualization follows:

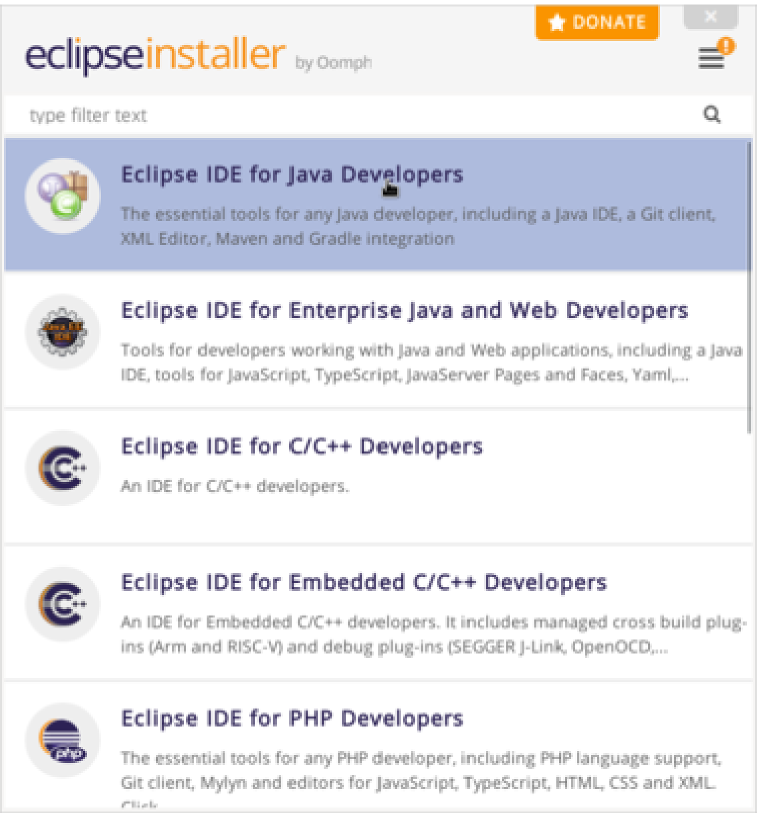


23.9 StarPU Eclipse Plugin

The StarPU Eclipse Plugin provides the ability to generate the different traces directly from the Eclipse IDE.

23.9.1 Eclipse Installation

Download the Eclipse installer from https://www.eclipse.org/downloads/packages/installer. When you run the installer, click on *Eclipse IDE for Java Developers* to start the installation process.



Eclipse IDE for Java Developers
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration

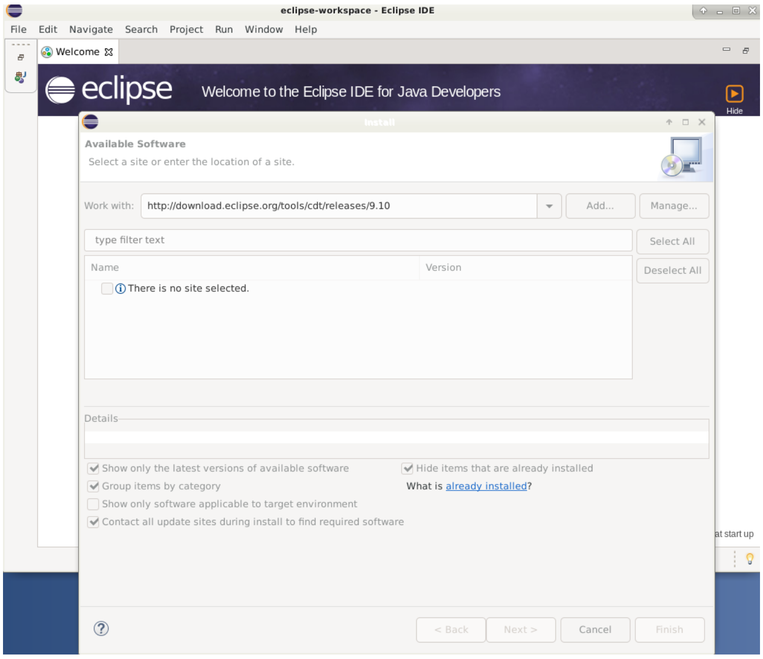
Eclipse IDE for Enterprise Java and Web Developers
Tools for developers working with Java and Web applications, including a Java IDE, tools for JavaScript, TypeScript, JavaServer Pages and Faces, Yaml,...

Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers.

Eclipse IDE for Embedded C/C++ Developers
An IDE for Embedded C/C++ developers. It includes managed cross build plug-ins (Arm and RISC-V) and debug plug-ins (SEGGER J-Link, OpenOCD,...

Eclipse IDE for PHP Developers
The essential tools for any PHP developer, including PHP language support, Git client, Mylyn and editors for JavaScript, TypeScript, HTML, CSS and XML.

To be able to develop C/C++ applications, you need to install the CDT plugin. To do so, go to the *Help* dropdown menu at the top of the Eclipse window, choose *Install New Software ....* In the new window, enter the URL http://download.eclipse.org/tools/cdt/releases/9.10 into the box *Work with* and press the return key.



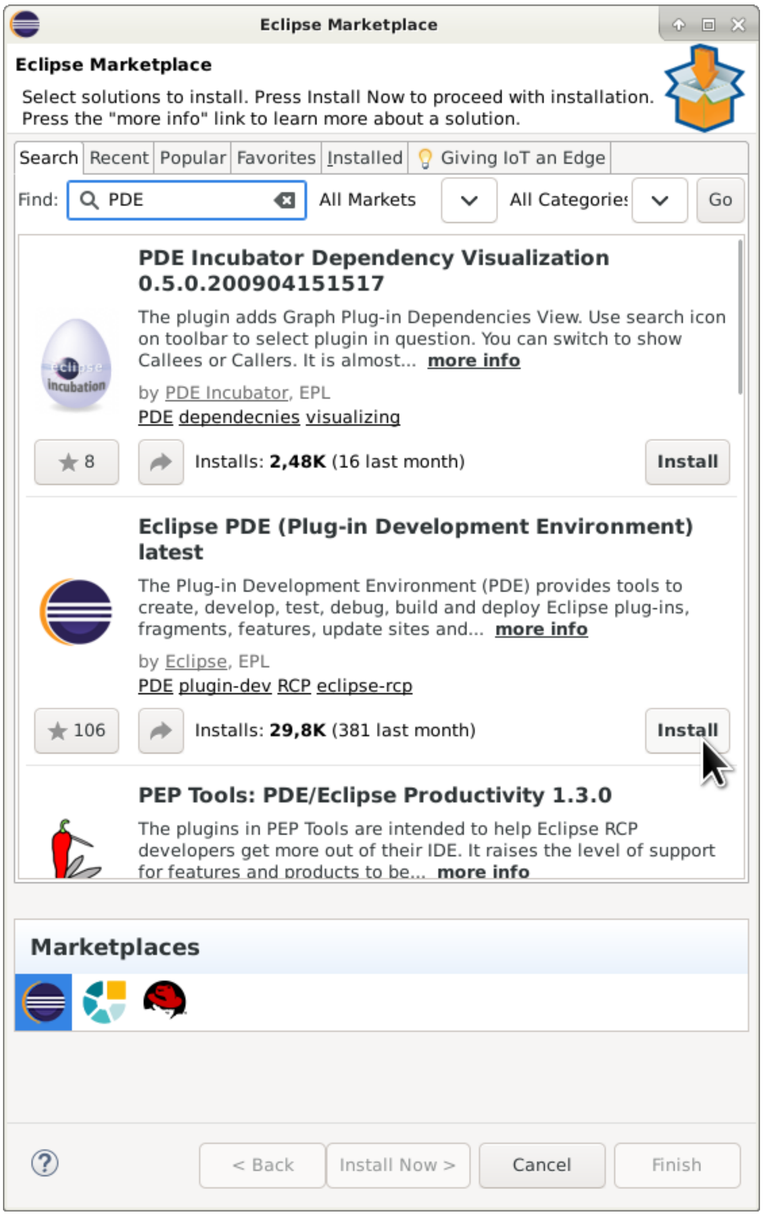
Available Software
Select a site or enter the location of a site.

Work with: http://download.eclipse.org/tools/cdt/releases/9.10

☐ There is no site selected.

☑ Show only the latest versions of available software
☑ Group items by category
☐ Show only software applicable to target environment
☑ Contact all update sites during install to find required software
☑ Hide items that are already installed
What is already installed?

You need then to select *CDT Main Features*, then click the button *Next* twice, accept the terms of the license, and click the button *Finish*. Eclipse will ask you to restart.
To be able to compile the plugin, you need to install the plugin development environment (PDE). To do so, go to the menu *Help*, choose *Eclipse Marketplace....* In the new window, enter *PDE* into the box *Find* and press the return key.



Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.

Find: PDE

PDE Incubator Dependency Visualization 0.5.0.200904151517

The plugin adds Graph Plug-in Dependencies View. Use search icon on toolbar to select plugin in question. You can switch to show Callees or Callers. It is almost... **more info**

by PDE Incubator, EPL
PDE dependecnies visualizing

★ 8 Installs: **2,48K** (16 last month)

Eclipse PDE (Plug-in Development Environment) latest

The Plug-in Development Environment (PDE) provides tools to create, develop, test, debug, build and deploy Eclipse plug-ins, fragments, features, update sites and... **more info**

by Eclipse, EPL
PDE plugin-dev RCP eclipse-rcp

★ 106 Installs: **29,8K** (381 last month)

PEP Tools: PDE/Eclipse Productivity 1.3.0

The plugins in PEP Tools are intended to help Eclipse RCP developers get more out of their IDE. It raises the level of support for features and products to be... **more info**

You can then click on the button *Install* of the *Eclipse PDE latest*. You may need to confirm the installation, then accept the terms of the license, and finally restart the Eclipse IDE.
The installation is now done.

23.9.2 StarPU Eclipse Plugin Compilation and Installation

StarPU can now be compiled and installed with its Eclipse plugin. To do so, you first need to configure StarPU with the option --enable-eclipse-plugin. The Eclipse IDE executable `eclipse` must be in your `PATH`.

```
export PATH=$HOME/usr/local/eclipse/java-2021-03/eclipse:$PATH
mkdir build
cd build
../configure --prefix=$HOME/usr/local/starpu --enable-eclipse-plugin
make
make install
```

The StarPU Eclipse plugin is installed in the directory `dropins`.

```
$ ls $HOME/usr/local/eclipse/java-2021-03/eclipse/dropins
StarPU_1.0.0.202105272056.jar
```

In the next section, we will show you how to use the plugin.

23.9.3 StarPU Eclipse Plugin Instruction

Once StarPU has been configured and installed with its Eclipse plugin, you first need to set up your environment for StarPU.

```
cd $HOME/usr/local/starpu
source ./bin/starpu_env
```

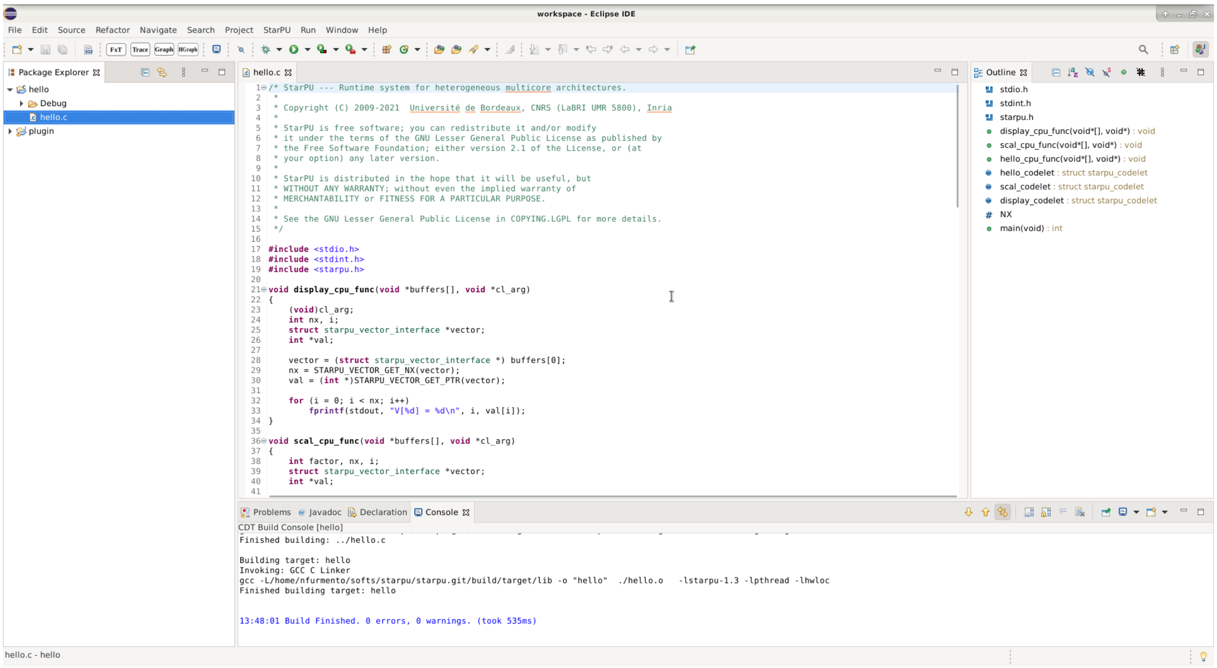
To generate traces from the application, it is necessary to set STARPU_FXT_TRACE to 1.

```
export STARPU_FXT_TRACE=1
```

The eclipse workspace together with an example is available in `lib/starpu/eclipse-plugin`.

```
cd ./lib/starpu/eclipse-plugin
eclipse -data workspace
```

You can then open the file `hello/hello.c`, and build the application by pressing `Ctrl-B`.

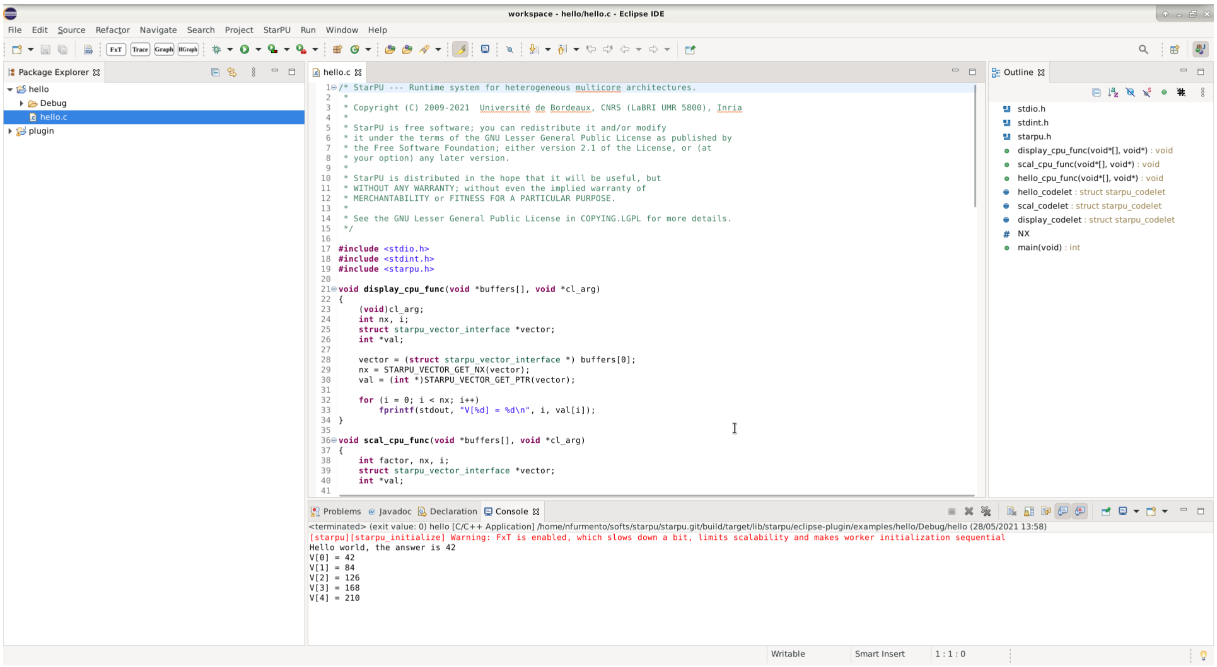

workspace - Eclipse IDE
File Edit Source Refactor Navigate Search Project StarPU Run Window Help
FxT Trace Graph HGraph
Package Explorer
hello
Debug
hello.c
plugin
hello.c
Outline
stdio.h
stdint.h
starpu.h
display_cpu_func(void*[], void*) : void
scal_cpu_func(void*[], void*) : void
hello_cpu_func(void*[], void*) : void
hello_codelet : struct starpu_codelet
scal_codelet : struct starpu_codelet
display_codelet : struct starpu_codelet
NX
main(void) : int
Problems Javadoc Declaration Console
hello.c - hello

```
/* StarPU --- Runtime system for heterogeneous multicore architectures.
 *
 * Copyright (C) 2009-2021  Université de Bordeaux, CNRS (LaBRI UMR 5800), Inria
 *
 * StarPU is free software; you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License as published by
 * the Free Software Foundation; either version 2.1 of the License, or (at
 * your option) any later version.
 *
 * StarPU is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 *
 * See the GNU Lesser General Public License in COPYING.LGPL for more details.
 */

#include <stdio.h>
#include <stdint.h>
#include <starpu.h>

void display_cpu_func(void *buffers[], void *cl_arg)
{
    (void)cl_arg;
    int nx, i;
    struct starpu_vector_interface *vector;
    int *val;

    vector = (struct starpu_vector_interface *) buffers[0];
    nx = STARPU_VECTOR_GET_NX(vector);
    val = (int *)STARPU_VECTOR_GET_PTR(vector);

    for (i = 0; i < nx; i++)
        fprintf(stdout, "V[%d] = %d\n", i, val[i]);
}

void scal_cpu_func(void *buffers[], void *cl_arg)
{
    int factor, nx, i;
    struct starpu_vector_interface *vector;
    int *val;

```

CDT Build Console [hello]

```
Finished building: ../hello.c

Building target: hello
Invoking: GCC C Linker
gcc -L/home/nfurmento/softs/starpu/starpu.git/build/target/lib -o "hello"  ./hello.o   -lstarpu-1.3 -lpthread -lhwloc
Finished building target: hello


13:48:01 Build Finished. 0 errors, 0 warnings. (took 535ms)
```

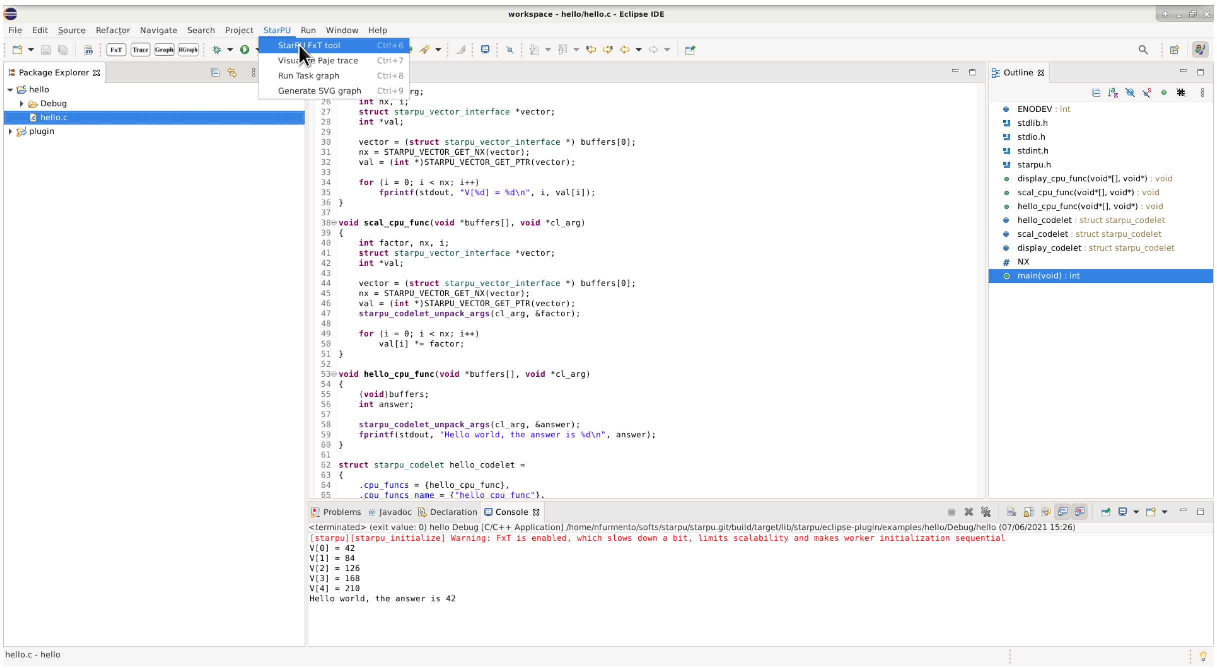
The application can now be executed.


workspace - hello/hello.c - Eclipse IDE
File Edit Source Refactor Navigate Search Project StarPU Run Window Help
FxT Trace Graph HGraph
Package Explorer
hello
Debug
hello.c
plugin
hello.c
Outline
stdio.h
stdint.h
starpu.h
display_cpu_func(void*[], void*) : void
scal_cpu_func(void*[], void*) : void
hello_cpu_func(void*[], void*) : void
hello_codelet : struct starpu_codelet
scal_codelet : struct starpu_codelet
display_codelet : struct starpu_codelet
NX
main(void) : int
Problems Javadoc Declaration Console
Writable
Smart Insert
1 : 1 : 0

```
/* StarPU --- Runtime system for heterogeneous multicore architectures.
 *
 * Copyright (C) 2009-2021  Université de Bordeaux, CNRS (LaBRI UMR 5800), Inria
 *
 * StarPU is free software; you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License as published by
 * the Free Software Foundation; either version 2.1 of the License, or (at
 * your option) any later version.
 *
 * StarPU is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 *
 * See the GNU Lesser General Public License in COPYING.LGPL for more details.
 */

#include <stdio.h>
#include <stdint.h>
#include <starpu.h>

void display_cpu_func(void *buffers[], void *cl_arg)
{
    (void)cl_arg;
    int nx, i;
    struct starpu_vector_interface *vector;
    int *val;

    vector = (struct starpu_vector_interface *) buffers[0];
    nx = STARPU_VECTOR_GET_NX(vector);
    val = (int *)STARPU_VECTOR_GET_PTR(vector);

    for (i = 0; i < nx; i++)
        fprintf(stdout, "V[%d] = %d\n", i, val[i]);
}

void scal_cpu_func(void *buffers[], void *cl_arg)
{
    int factor, nx, i;
    struct starpu_vector_interface *vector;
    int *val;

```

```
<terminated> (exit value: 0) hello [C/C++ Application] /home/nfurmento/softs/starpu/starpu.git/build/target/lib/starpu/eclipse-plugin/examples/hello/Debug/hello (28/05/2021 13:58)
[starpu][starpu_initialize] Warning: FxT is enabled, which slows down a bit, limits scalability and makes worker initialization sequential
Hello world, the answer is 42
V[0] = 42
V[1] = 84
V[2] = 126
V[3] = 168
V[4] = 210
```

After executing the C/C++ StarPU application, one can use the StarPU plugin to generate and visualise the task graph of the application. The StarPU plugin eclipse is either available through the icons in the upper toolbar, or from the dropdown menu `StarPU`.

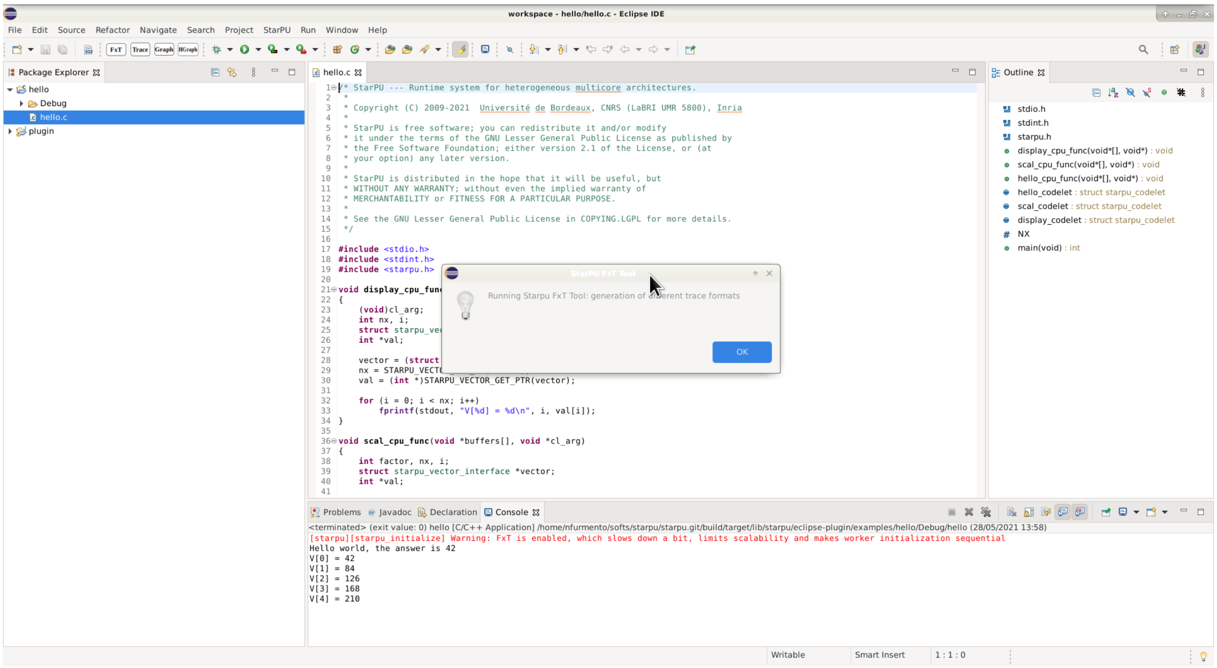

workspace - hello/hello.c - Eclipse IDE
File Edit Source Refactor Navigate Search Project StarPU Run Window Help
FxT Trace Graph HGraph
StarPU FxT tool Ctrl+6
Visualize Paje trace Ctrl+7
Run Task graph Ctrl+8
Generate SVG graph Ctrl+9
Package Explorer
hello
Debug
hello.c
plugin
Outline
ENODEV : int
stdlib.h
stdio.h
stdint.h
starpu.h
display_cpu_func(void*[], void*) : void
scal_cpu_func(void*[], void*) : void
hello_cpu_func(void*[], void*) : void
hello_codelet : struct starpu_codelet
scal_codelet : struct starpu_codelet
display_codelet : struct starpu_codelet
NX
main(void) : int
Problems Javadoc Declaration Console
hello.c - hello

```
rg;
    int nx, i;
    struct starpu_vector_interface *vector;
    int *val;

    vector = (struct starpu_vector_interface *) buffers[0];
    nx = STARPU_VECTOR_GET_NX(vector);
    val = (int *)STARPU_VECTOR_GET_PTR(vector);

    for (i = 0; i < nx; i++)
        fprintf(stdout, "V[%d] = %d\n", i, val[i]);
}

void scal_cpu_func(void *buffers[], void *cl_arg)
{
    int factor, nx, i;
    struct starpu_vector_interface *vector;
    int *val;

    vector = (struct starpu_vector_interface *) buffers[0];
    nx = STARPU_VECTOR_GET_NX(vector);
    val = (int *)STARPU_VECTOR_GET_PTR(vector);
    starpu_codelet_unpack_args(cl_arg, &factor);

    for (i = 0; i < nx; i++)
        val[i] *= factor;
}

void hello_cpu_func(void *buffers[], void *cl_arg)
{
    (void)buffers;
    int answer;

    starpu_codelet_unpack_args(cl_arg, &answer);
    fprintf(stdout, "Hello world, the answer is %d\n", answer);
}

struct starpu_codelet hello_codelet =
{
    .cpu_funcs = {hello_cpu_func},
    .cpu_funcs_name = {"hello_cpu_func"},
```

```
<terminated> (exit value: 0) hello Debug [C/C++ Application] /home/nfurmento/softs/starpu/starpu.git/build/target/lib/starpu/eclipse-plugin/examples/hello/Debug/hello (07/06/2021 15:26)
[starpu][starpu_initialize] Warning: FxT is enabled, which slows down a bit, limits scalability and makes worker initialization sequential
V[0] = 42
V[1] = 84
V[2] = 126
V[3] = 168
V[4] = 210
Hello world, the answer is 42
```

To start, one first need to run the StarPU FxT tool, either through the `FxT` icon of the toolbar, or from the menu `StarPU` / `StarPU FxT Tool`. This will call the tool `starpu_fxt_tool` to generate traces for your application execution.

A message dialog box is displayed to confirm the generation of the different traces.

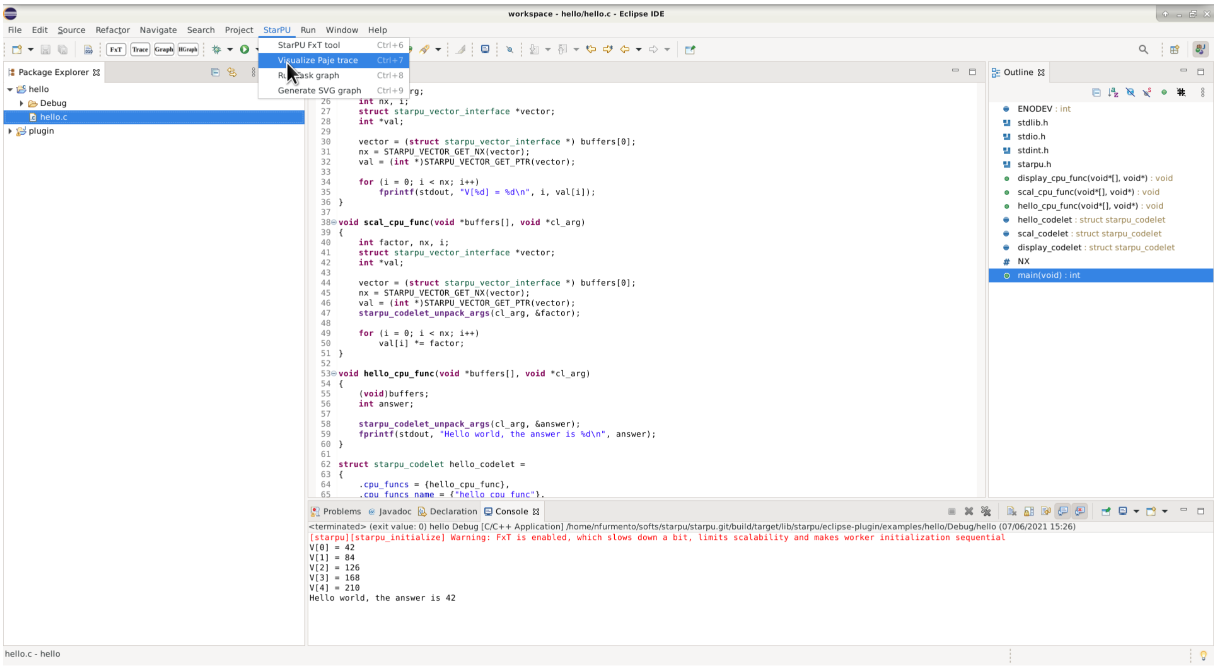

workspace - hello/hello.c - Eclipse IDE
File Edit Source Refactor Navigate Search Project StarPU Run Window Help
FxT Trace Graph HGraph
Package Explorer
hello
Debug
hello.c
plugin
hello.c
Outline
stdio.h
stdint.h
starpu.h
display_cpu_func(void*[], void*) : void
scal_cpu_func(void*[], void*) : void
hello_cpu_func(void*[], void*) : void
hello_codelet : struct starpu_codelet
scal_codelet : struct starpu_codelet
display_codelet : struct starpu_codelet
NX
main(void) : int
StarPU FxT Tool
OK
Problems Javadoc Declaration Console
Writable
Smart Insert
1 : 1 : 0

```
/* StarPU --- Runtime system for heterogeneous multicore architectures.
 *
 * Copyright (C) 2009-2021  Université de Bordeaux, CNRS (LaBRI UMR 5800), Inria
 *
 * StarPU is free software; you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License as published by
 * the Free Software Foundation; either version 2.1 of the License, or (at
 * your option) any later version.
 *
 * StarPU is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 *
 * See the GNU Lesser General Public License in COPYING.LGPL for more details.
 */

#include <stdio.h>
#include <stdint.h>
#include <starpu.h>

void display_cpu_func
{
    (void)cl_arg;
    int nx, i;
    struct starpu_ve
    int *val;

    vector = (struct
    nx = STARPU_VECT
    val = (int *)STARPU_VECTOR_GET_PTR(vector);

    for (i = 0; i < nx; i++)
        fprintf(stdout, "V[%d] = %d\n", i, val[i]);
}

void scal_cpu_func(void *buffers[], void *cl_arg)
{
    int factor, nx, i;
    struct starpu_vector_interface *vector;
    int *val;

```

Running Starpu FxT Tool: generation of different trace formats

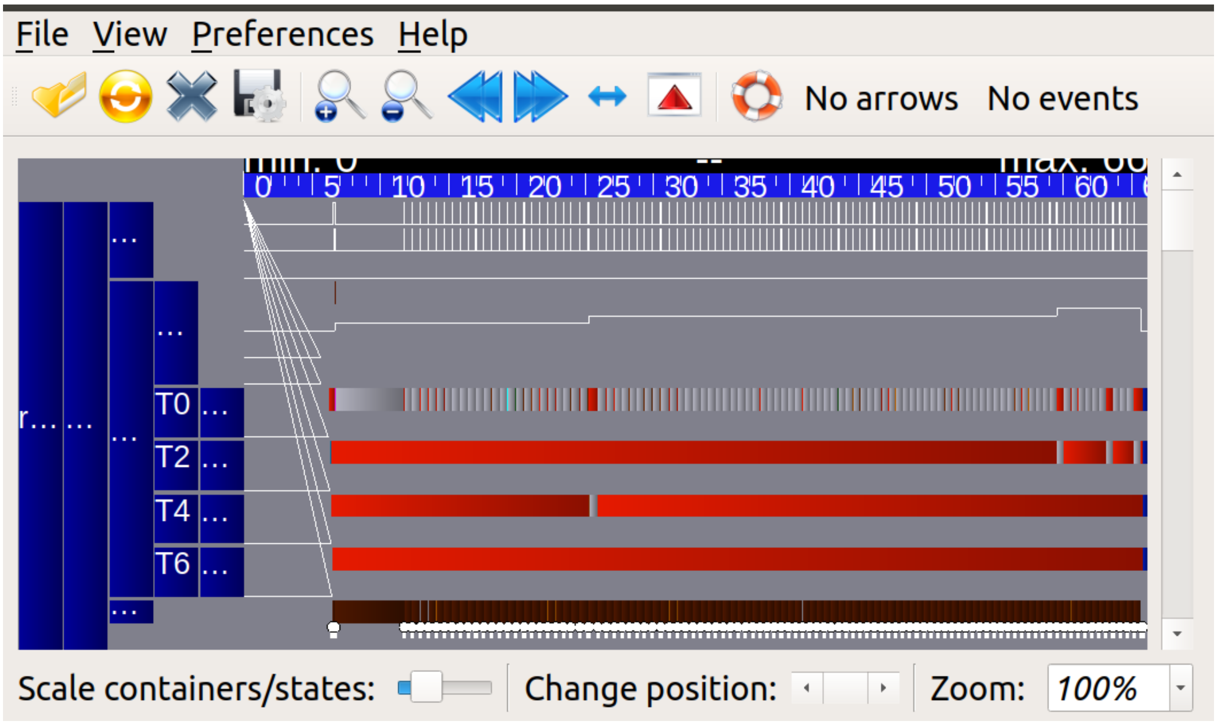
```
<terminated> (exit value: 0) hello [C/C++ Application] /home/nfurmento/softs/starpu/starpu.git/build/target/lib/starpu/eclipse-plugin/examples/hello/Debug/hello (28/05/2021 13:58)
[starpu][starpu_initialize] Warning: FxT is enabled, which slows down a bit, limits scalability and makes worker initialization sequential
Hello world, the answer is 42
V[0] = 42
V[1] = 84
V[2] = 126
V[3] = 168
V[4] = 210
```

One of the generated files is a Paje trace which can be viewed with ViTE, a trace explorer. To open and visualise the file `paje.trace` with ViTE, one can select the second command of the StarPU menu, which is named `Generate Paje Trace`, or click on the second icon named `Trace` in the toolbar.

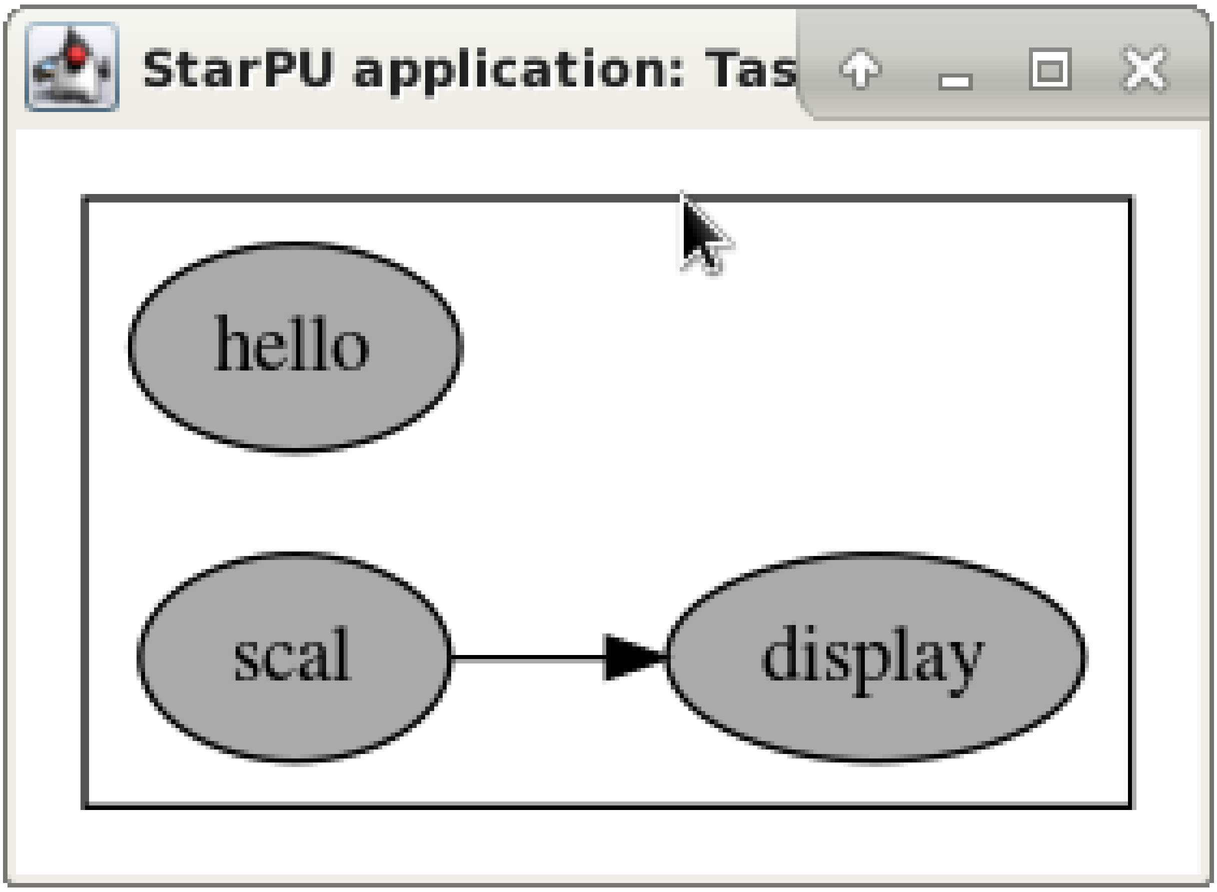

workspace - hello/hello.c - Eclipse IDE
File Edit Source Refactor Navigate Search Project StarPU Run Window Help
StarPU FxT tool Ctrl+6
Visualize Paje trace Ctrl+7
Run task graph Ctrl+8
Generate SVG graph Ctrl+9
Package Explorer
hello
Debug
hello.c
plugin
Outline
ENODEV : int
stdlib.h
stdio.h
stdint.h
starpu.h
display_cpu_func(void*[], void*) : void
scal_cpu_func(void*[], void*) : void
hello_cpu_func(void*[], void*) : void
hello_codelet : struct starpu_codelet
scal_codelet : struct starpu_codelet
display_codelet : struct starpu_codelet
NX
main(void) : int
Problems Javadoc Declaration Console
hello.c - hello

```
    int nx, i;
    struct starpu_vector_interface *vector;
    int *val;

    vector = (struct starpu_vector_interface *) buffers[0];
    nx = STARPU_VECTOR_GET_NX(vector);
    val = (int *)STARPU_VECTOR_GET_PTR(vector);

    for (i = 0; i < nx; i++)
        fprintf(stdout, "V[%d] = %d\n", i, val[i]);
}

void scal_cpu_func(void *buffers[], void *cl_arg)
{
    int factor, nx, i;
    struct starpu_vector_interface *vector;
    int *val;

    vector = (struct starpu_vector_interface *) buffers[0];
    nx = STARPU_VECTOR_GET_NX(vector);
    val = (int *)STARPU_VECTOR_GET_PTR(vector);
    starpu_codelet_unpack_args(cl_arg, &factor);

    for (i = 0; i < nx; i++)
        val[i] *= factor;
}

void hello_cpu_func(void *buffers[], void *cl_arg)
{
    (void)buffers;
    int answer;

    starpu_codelet_unpack_args(cl_arg, &answer);
    fprintf(stdout, "Hello world, the answer is %d\n", answer);
}

struct starpu_codelet hello_codelet =
{
    .cpu_funcs = {hello_cpu_func},
    .cpu_funcs_name = {"hello_cpu_func"},
```

```
<terminated> (exit value: 0) hello Debug [C/C++ Application] /home/nfurmento/softs/starpu/starpu.git/build/target/lib/starpu/eclipse-plugin/examples/hello/Debug/hello (07/06/2021 15:26)
[starpu][starpu_initialize] Warning: FxT is enabled, which slows down a bit, limits scalability and makes worker initialization sequential
V[0] = 42
V[1] = 84
V[2] = 126
V[3] = 168
V[4] = 210
Hello world, the answer is 42
```

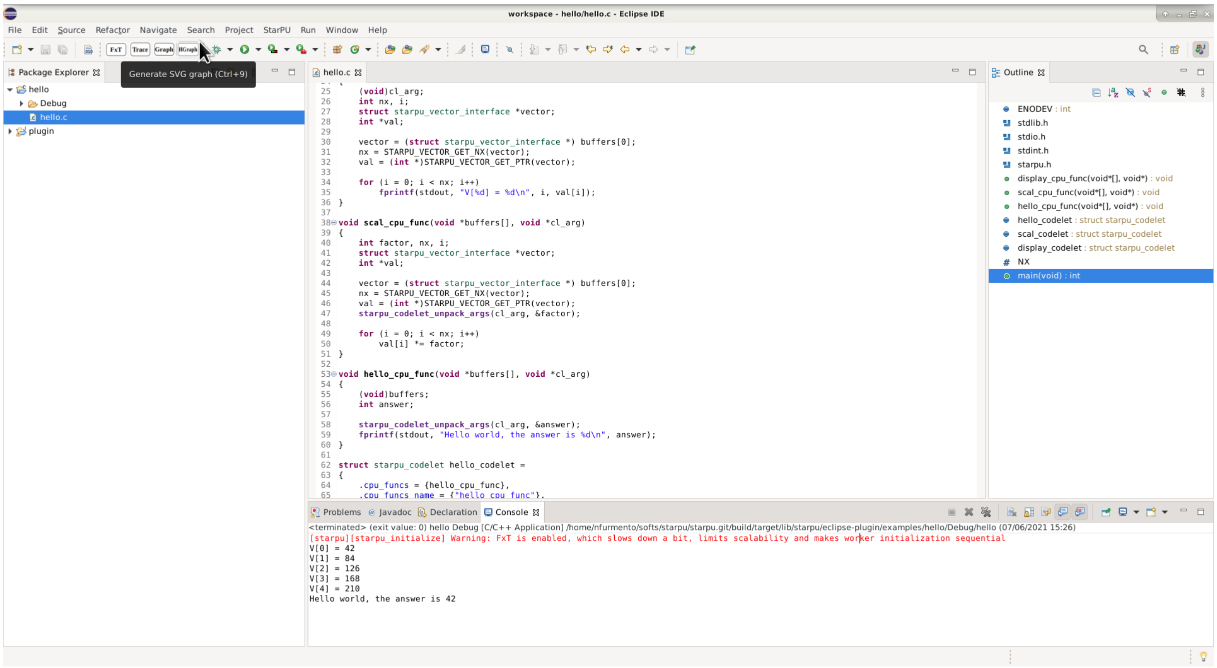


Another generated trace file is a task graph described using the DOT language. It is possible to get a graphical output of the graph by calling the `graphviz library`. To do this, one can click on the third command of StarPU menu. A task graph of the application in the `png` format is then generated.



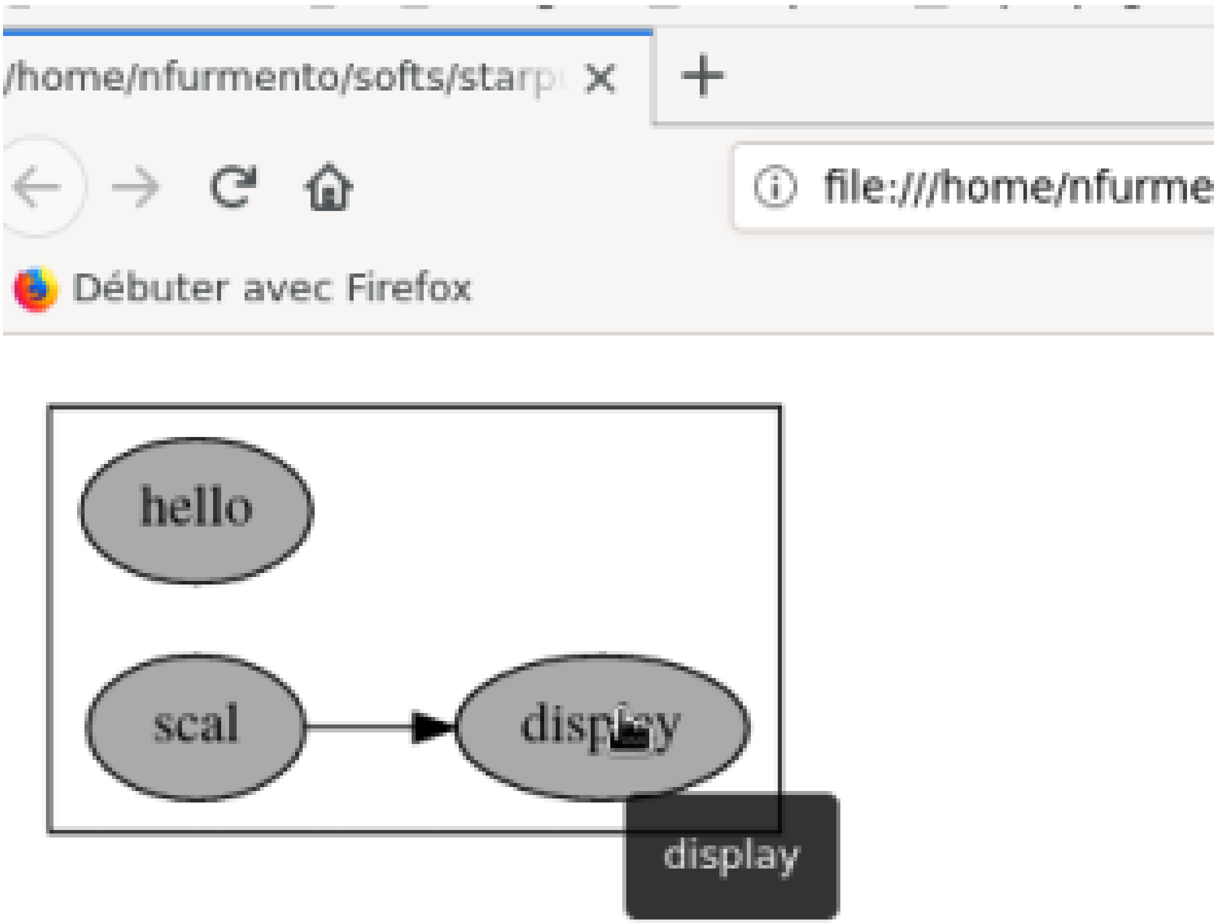
In StarPU eclipse plugin, one can display the graph task directly from eclipse, or through a web browser. To do this, there is another command named `Generate SVG graph` in the StarPU menu or HGraph in the toolbar of eclipse.

From the HTML file, you can see the graph task, and by clicking on a task name, it will open the C file in which the task submission was called (if you have an editor which understands the syntax `href="file.c#123"`).


workspace - hello/hello.c - Eclipse IDE
File
Edit
Source
Refactor
Navigate
Search
Project
StarPU
Run
Window
Help
FxT
Trace
Graph
HGraph
Package Explorer
Generate SVG graph (Ctrl+9)
hello
Debug
hello.c
plugin
hello.c
Outline
ENODEV : int
stdlib.h
stdio.h
stdint.h
starpu.h
display_cpu_func(void*[], void*) : void
scal_cpu_func(void*[], void*) : void
hello_cpu_func(void*[], void*) : void
hello_codelet : struct starpu_codelet
scal_codelet : struct starpu_codelet
display_codelet : struct starpu_codelet
NX
main(void) : int
Problems
Javadoc
Declaration
Console

```
    (void)cl_arg;
    int nx, i;
    struct starpu_vector_interface *vector;
    int *val;

    vector = (struct starpu_vector_interface *) buffers[0];
    nx = STARPU_VECTOR_GET_NX(vector);
    val = (int *)STARPU_VECTOR_GET_PTR(vector);

    for (i = 0; i < nx; i++)
        fprintf(stdout, "V[%d] = %d\n", i, val[i]);
}

void scal_cpu_func(void *buffers[], void *cl_arg)
{
    int factor, nx, i;
    struct starpu_vector_interface *vector;
    int *val;

    vector = (struct starpu_vector_interface *) buffers[0];
    nx = STARPU_VECTOR_GET_NX(vector);
    val = (int *)STARPU_VECTOR_GET_PTR(vector);
    starpu_codelet_unpack_args(cl_arg, &factor);

    for (i = 0; i < nx; i++)
        val[i] *= factor;
}

void hello_cpu_func(void *buffers[], void *cl_arg)
{
    (void)buffers;
    int answer;

    starpu_codelet_unpack_args(cl_arg, &answer);
    fprintf(stdout, "Hello world, the answer is %d\n", answer);
}

struct starpu_codelet hello_codelet =
{
    .cpu_funcs = {hello_cpu_func},
    .cpu_funcs_name = {"hello_cpu_func"},
```

```
<terminated> (exit value: 0) hello Debug [C/C++ Application] /home/nfurmento/softs/starpu/starpu.git/build/target/lib/starpu/eclipse-plugin/examples/hello/Debug/hello (07/06/2021 15:26)
[starpu][starpu_initialize] Warning: FxT is enabled, which slows down a bit, limits scalability and makes worker initialization sequential
V[0] = 42
V[1] = 84
V[2] = 126
V[3] = 168
V[4] = 210
Hello world, the answer is 42
```



23.10 Memory Feedback

It is possible to enable memory statistics. To do so, you need to pass the option --enable-memory-stats when running `configure`. It is then possible to call the function starpu_data_display_memory_stats() to display statistics about the current data handles registered within StarPU.
Moreover, statistics will be displayed at the end of the execution on data handles which have not been cleared out. This can be disabled by setting the environment variable STARPU_MEMORY_STATS to `0`.
For example, by adding a call to the function starpu_data_display_memory_stats() in the fblock example before unpartitioning the data, one will get something similar to:

```
$ STARPU_MEMORY_STATS=1 ./examples/filters/fblock
...
#---------------------
Memory stats :
#-------
Data on Node #2
#-----
Data : 0x5562074e8670
Size : 144

#--
Data access stats
```

```
/!\ Work Underway
Node #0
        Direct access : 0
        Loaded (Owner) : 0
        Loaded (Shared) : 0
        Invalidated (was Owner) : 1

Node #2
        Direct access : 0
        Loaded (Owner) : 1
        Loaded (Shared) : 0
        Invalidated (was Owner) : 0

#-------
Data on Node #3
#-----
Data : 0x5562074e9338
Size : 96

#--
Data access stats
/!\ Work Underway
Node #0
        Direct access : 0
        Loaded (Owner) : 0
        Loaded (Shared) : 0
        Invalidated (was Owner) : 1

Node #3
        Direct access : 0
        Loaded (Owner) : 1
        Loaded (Shared) : 0
        Invalidated (was Owner) : 0


#---------------------
...
```

23.11 Data Statistics

Different data statistics can be displayed at the end of the execution of the application. To enable them, you need to define the environment variable STARPU_ENABLE_STATS. When calling starpu_shutdown() various statistics will be displayed, execution, MSI cache statistics, allocation cache statistics, and data transfer statistics. The display can be disabled by setting the environment variable STARPU_STATS to `0`. If the environment variable STARPU_BUS_STATS is defined, you can call starpu_profiling_bus_helper_display_summary() to display statistics about the bus. If the environment variable STARPU_WORKER_STATS is defined, you can call starpu_profiling_worker_helper_display_summary() to display statistics about the workers. You can also call starpu_display_stats() which call both starpu_profiling_bus_helper_display_summary() and starpu_profiling_worker_helper_display_summary() at the same time.

```
$ ./examples/cholesky/cholesky_tag
Computation took (in ms)
518.16
Synthetic GFlops : 44.21
#---------------------
MSI cache stats :
TOTAL MSI stats hit 1622 (66.23 %)      miss 827 (33.77 %)
...
```

```
$ STARPU_STATS=0 ./examples/cholesky/cholesky_tag
Computation took (in ms)
518.16
Synthetic GFlop/s : 44.21
```

23.12 Tracing MPI applications

When an MPI execution is traced, especially if the execution is on several nodes, clock synchronization issues can appear. One may notice them mainly on communications (they are received before they are sent, for instance).

Each processor can call the function starpu_profiling_set_id() to set the ID used for the profiling trace filename. This function can be useful when executing an MPI program on several nodes, as it enables each processor to set a unique ID that helps to differentiate its trace file from the files generated by other processors. By doing this, it becomes easier to analyze and compare the profiling results of each processor separately, which is particularly helpful for large-scale parallel applications.

By default, StarPU does two MPI barriers with all MPI processes: one at the beginning of the application execution and one at the end. Then, `starpu_fxt_tool` considers all processes leave the barriers at the exact same time, which makes two points for time synchronization between MPI processes.

However, a simple MPI barrier can be not precise enough, because the assumption *all processes leave the barriers at the exact same time* is in reality false. To have a more precise barrier, one may use the `mpi_sync_clocks library` (automatically provided when StarPU is built with NewMadeleine, but it can also be used with other MPI libraries). It provides a *synchronized* barrier, which aims at actually releasing all processes at the exact same time. Unfortunately, the gained precision costs some time (several seconds per barrier), that is why one can disable this precise synchronization with the environment variable STARPU_MPI_TRACE_SYNC_CLOCKS set to `0`, and use the faster MPI barrier instead.

23.13 Verbose Traces

Traces can also be inspected by hand by using the tool `fxt_print`, for instance:

```
$ fxt_print -o -f /tmp/prof_file_something
```

Timings are in nanoseconds (while timings as seen in ViTE are in milliseconds).

Part V

StarPU FAQ

Chapter 24

Organization

This part explains how to better tune your application to achieve good performance, and also how to fix some difficulties you may encounter while implementing your applications.

- We give a list of features in Chapter Check List When Performance Are Not There which should be checked to improve performances of your applications.
- There are some frequently asked questions in Chapter Frequently Asked Questions that may help you to solve your problems.

If you have problems that cannot be solved, please contact us.

Chapter 25

Check List When Performance Are Not There

TODO: improve!
To achieve good performance, we give below a list of features which should be checked.
For a start, you can use Offline Performance Tools to get a Gantt chart which will show roughly where time is spent, and focus correspondingly.

25.1 Check Task Size

Make sure that your tasks are not too small, as the StarPU runtime overhead may not be negligible. As explained in Task Size Overhead, you can run the script `tasks_size_overhead.sh` to get an idea of the scalability of tasks depending on their duration (in µs), on your own system.
Typically, 10µs-ish tasks are definitely too small, the CUDA overhead itself is much bigger than this.
1ms-ish tasks may be a good start, but will not necessarily scale to many dozens of cores, so it's better to try to get 10ms-ish tasks.
It may be useful to dedicate a whole core to the main thread, so it can spend its time on submitting tasks, by setting the STARPU_MAIN_THREAD_BIND environment variable to 1.
Tasks durations can easily be observed when performance models are defined (see Performance Model Example) by using the tools `starpu_perfmodel_plot` or `starpu_perfmodel_display` (see Performance Of Codelets)
When using parallel tasks, the problem is even worse since StarPU has to synchronize the tasks execution.

25.2 Configuration Which May Improve Performance

If you do not plan to use support for GPUs or out-of-core, i.e. not use StarPU's ability to manage data coherency between several memory nodes, the `configure` option --enable-maxnodes=1 allows to considerably reduce StarPU's memory management overhead.
The `configure` option --enable-fast disables all assertions. This makes StarPU more performant for tiny tasks by disabling all sanity checks. Only use this for measurements and production, not for development, since this will drop all basic checks.

25.3 Data Related Features Which May Improve Performance

As can be seen in States in Gantt Diagram, if the application has a lot of different kinds of sizes of data, StarPU will end up freeing/reallocating data on GPU to accomodate for the different sizes. It can be very effective to round the allocated size up a bit by e.g. 10% (e.g. 11MB for all data sizes between 10MB and 11MB) so that StarPU will be able to reuse buffers of the same size for data with similar but not exactly same size. This can be registered by using starpu_matrix_data_register_allocsize(), starpu_vector_data_register_allocsize() so that StarPU records both the rounded-up data size, and the actual size used for computation.
link to Data Management
link to Data Prefetch

25.4 Task Related Features Which May Improve Performance

link to Task Granularity
link to Task Submission
link to Task Priorities

25.5 Scheduling Related Features Which May Improve Performance

link to Task Scheduling Policies
link to Task Distribution Vs Data Transfer
link to Energy-based Scheduling
link to Static Scheduling

25.6 CUDA-specific Optimizations

For proper overlapping of asynchronous GPU data transfers, data has to be pinned by CUDA. Data allocated with starpu_malloc() is always properly pinned. If the application registers to StarPU some data which has not been allocated with starpu_malloc(), starpu_memory_pin() should be called to pin the data memory. Otherwise, the "Asynchronous copy submission" parts of the execution traces (see States in Gantt Diagram) will show the synchronous inefficiency.

Note that CUDA pinning/unpinning takes a long time, so for e.g. temporary data, it is much more efficient to use a StarPU temporary data (see Temporary Data), that StarPU can reuse and thus avoid the pin/unpin cost.

Due to CUDA limitations, StarPU will have a hard time overlapping its own communications and the codelet computations if the application does not use a dedicated CUDA stream for its computations instead of the default stream, which synchronizes all operations of the GPU. The function starpu_cuda_get_local_stream() returns a stream which can be used by all CUDA codelet operations to avoid this issue. For instance:

```
func «<grid,block,0,starpu_cuda_get_local_stream()»> (foo, bar);
cudaError_t status = cudaGetLastError();
if (status != cudaSuccess) STARPU_CUDA_REPORT_ERROR(status);
cudaStreamSynchronize(starpu_cuda_get_local_stream());
```

as well as the use of `cudaMemcpyAsync()`, etc. for each CUDA operation one needs to use a version that takes a stream parameter.

If the kernel uses its own non-default stream, one can synchronize this stream with the StarPU-provided stream this way:

```
cudaEvent_t event;
call_kernel_with_its_own_stream()
cudaEventCreateWithFlags(&event, cudaEventDisableTiming);
cudaEventRecord(event, get_kernel_stream());
cudaStreamWaitEvent(starpu_cuda_get_local_stream(), event, 0);
cudaEventDestroy(event);
```

This code makes the StarPU-provided stream wait for a new event, which will be triggered by the completion of the kernel.

Unfortunately, some CUDA libraries do not have stream variants of kernels. This will seriously lower the potential for overlapping. If some CUDA calls are made without specifying this local stream, synchronization needs to be explicit with cudaDeviceSynchronize() around these calls, to make sure that they get properly synchronized with the calls using the local stream. Notably, `cudaMemcpy()` and `cudaMemset()` are actually asynchronous and need such explicit synchronization! Use `cudaMemcpyAsync()` and `cudaMemsetAsync()` instead.

Calling starpu_cublas_init() will ensure StarPU to properly call the CUBLAS library functions, and starpu_cublas_shutdown() will synchronously deinitialize the CUBLAS library on every CUDA device. Some libraries like Magma may however change the current stream of CUBLAS v1, one then has to call starpu_cublas_set_stream() at the beginning of the codelet to make sure that CUBLAS is really using the proper stream. When using CUBLAS v2, starpu_cublas_get_local_handle() can be called to queue CUBLAS kernels with the proper configuration.

Similarly, calling starpu_cusparse_init() makes StarPU create CUSPARSE handles on each CUDA device, starpu_cusparse_get_local_handle() can then be used to queue CUSPARSE kernels with the proper configuration. starpu_cusparse_shutdown() will synchronously deinitialize the CUSPARSE library on every CUDA device.

Similarly, calling starpu_cusolver_init() makes StarPU create CUSOLVER handles on each CUDA device, starpu_cusolverDn_get_local_handle(), starpu_cusolverSp_get_local_handle(), starpu_cusolverRf_get_local_handle(), can then be used to queue CUSOLVER kernels with the proper configuration. starpu_cusolver_shutdown() can be used to clear these handles. It is useful to use a STARPU_SCRATCH buffer whose size was set to the amount returned by `cusolver*Spotrf_bufferSize`. An example can be seen in `examples/cholesky`

If the kernel can be made to only use this local stream or other self-allocated streams, i.e. the whole kernel submission can be made asynchronous, then one should enable asynchronous execution of the kernel. This means setting the flag STARPU_CUDA_ASYNC in the corresponding field starpu_codelet::cuda_flags, and dropping the `cudaStreamSynchronize()` call at the end of the `cuda_func` function, so that it returns immediately after having queued the kernel to the local stream. That way, StarPU will be able to submit and complete data transfers while kernels are executing, instead of only at each kernel submission. The kernel just has to make sure that StarPU can use the local stream to synchronize with the kernel startup and completion.
Using the flag STARPU_CUDA_ASYNC also permits to enable concurrent kernel execution, on cards which support it (Kepler and later, notably). This is enabled by setting the environment variable STARPU_NWORKER_PER_CUDA to the number of kernels to be executed concurrently. This is useful when kernels are small and do not feed the whole GPU with threads to run.
Concerning memory allocation, you should really not use `cudaMalloc()`/ `cudaFree()` within the kernel, since `cudaFree()` introduces way too many synchronizations within CUDA itself. You should instead add a parameter to the codelet with the STARPU_SCRATCH mode access. You can then pass to the task a handle registered with the desired size but with the `NULL` pointer, the handle can even be shared between tasks, StarPU will allocate per-task data on the fly before task execution, and reuse the allocated data between tasks.
See `examples/pi/pi_redux.c` for an example of use.

25.7 OpenCL-specific Optimizations

If the kernel can be made to only use the StarPU-provided command queue or other self-allocated queues, i.e. the whole kernel submission can be made asynchronous, then one should enable asynchronous execution of the kernel. This means setting the flag STARPU_OPENCL_ASYNC in the corresponding field starpu_codelet::opencl_flags and dropping the `clFinish()` and starpu_opencl_collect_stats() calls at the end of the kernel, so that it returns immediately after having queued the kernel to the provided queue. That way, StarPU will be able to submit and complete data transfers while kernels are executing, instead of only at each kernel submission. The kernel just has to make sure that StarPU can use the command queue it has provided to synchronize with the kernel startup and completion.

25.8 Detecting Stuck Conditions

It may happen that StarPU does not make progress for a long period of time. It may be due to contention inside StarPU, but it may also be an external problem, such as a stuck MPI or CUDA driver.
`export STARPU_WATCHDOG_TIMEOUT=10000` (STARPU_WATCHDOG_TIMEOUT)
allows making StarPU print an error message whenever StarPU does not terminate any task for 10ms, but lets the application continue normally. In addition to that,
`export STARPU_WATCHDOG_CRASH=1` (STARPU_WATCHDOG_CRASH)
raises `SIGABRT` in this condition, thus allowing to catch the situation in `gdb`.
It can also be useful to type `handle SIGABRT nopass` in `gdb` to be able to let the process continue, after inspecting the state of the process.

25.9 How to Limit Memory Used By StarPU And Cache Buffer Allocations

By default, StarPU makes sure to use at most 90% of the memory of GPU devices, moving data in and out of the device as appropriate, as well as using prefetch and write-back optimizations.
The environment variables STARPU_LIMIT_CUDA_MEM, STARPU_LIMIT_CUDA_devid_MEM, STARPU_LIMIT_OPENCL_MEM, and STARPU_LIMIT_OPENCL_devid_MEM can be used to control how much (in MiB) of the GPU device memory should be used at most by StarPU (the default value is to use 90% of the available memory).
By default, the usage of the main memory is not limited, as the default mechanisms do not provide means to evict main memory when it gets too tight. This also means that by default, StarPU will not cache buffer allocations in main memory, since it does not know how much of the system memory it can afford.
The environment variable STARPU_LIMIT_CPU_MEM can be used to specify how much (in MiB) of the main memory should be used at most by StarPU for buffer allocations. This way, StarPU will be able to cache buffer allocations (which can be a real benefit if a lot of buffers are involved, or if allocation fragmentation can become a problem), and when using Out Of Core, StarPU will know when it should evict data out to the disk.

It should be noted that by default only buffer allocations automatically done by StarPU are accounted here, i.e. allocations performed through starpu_malloc_on_node() which are used by the data interfaces (matrix, vector, etc.). This does not include allocations performed by the application through e.g. malloc(). It does not include allocations performed through starpu_malloc() either, only allocations performed explicitly with the flag STARPU_MALLOC_COUNT, i.e. by calling

```
starpu_malloc_flags(STARPU_MALLOC_COUNT)
```

are taken into account. And starpu_free_flags() can be called to free the memory that was previously allocated with starpu_malloc_flags(). If the application wants to make StarPU aware of its own allocations, so that StarPU knows precisely how much data is allocated, and thus when to evict allocation caches or data out to the disk, starpu_memory_allocate() can be used to specify an amount of memory to be accounted for. starpu_memory_deallocate() can be used to account freed memory back. Those can for instance be used by data interfaces with dynamic data buffers: instead of using starpu_malloc_on_node(), they would dynamically allocate data with `malloc()/realloc()`, and notify StarPU of the delta by calling starpu_memory_allocate() and starpu_memory_deallocate(). By default, the memory management system uses a set of default flags for each node when allocating memory. starpu_malloc_on_node_set_default_flags() can be used to modify these default flags on a specific node.

starpu_memory_get_total() and starpu_memory_get_available() can be used to get an estimation of how much memory is available. starpu_memory_wait_available() can also be used to block until an amount of memory becomes available, but it may be preferable to call

```
starpu_memory_allocate(STARPU_MEMORY_WAIT)
```

to reserve this amount immediately.

25.10 How To Reduce The Memory Footprint Of Internal Data Structures

It is possible to reduce the memory footprint of the task and data internal structures of StarPU by describing the shape of your machine and/or your application when calling `configure`.

To reduce the memory footprint of the data internal structures of StarPU, one can set the `configure` parameters --enable-maxcpus, --enable-maxnumanodes, --enable-maxcudadev, --enable-maxopencldev and --enable-maxnodes to give StarPU the architecture of the machine it will run on, thus tuning the size of the structures to the machine.

To reduce the memory footprint of the task internal structures of StarPU, one can set the `configure` parameter --enable-maxbuffers to give StarPU the maximum number of buffers that a task can use during an execution. For example, in the Cholesky factorization (dense linear algebra application), the GEMM task uses up to 3 buffers, so it is possible to set the maximum number of task buffers to 3 to run a Cholesky factorization on StarPU.

The size of the various structures of StarPU can be printed by `tests/microbenchs/display_structures_size`.

It is also often useless to submit **all** the tasks at the same time. Task submission can be blocked when a reasonable given number of tasks have been submitted, by setting the environment variables STARPU_LIMIT_MIN_SUBMITTED_TASKS and STARPU_LIMIT_MAX_SUBMITTED_TASKS.

```
export STARPU_LIMIT_MAX_SUBMITTED_TASKS=10000
export STARPU_LIMIT_MIN_SUBMITTED_TASKS=9000
```

will make StarPU block submission when 10000 tasks are submitted, and unblock submission when only 9000 tasks are still submitted, i.e. 1000 tasks have completed among the 10000 which were submitted when submission was blocked. Of course this may reduce parallelism if the threshold is set too low. The precise balance depends on the application task graph.

These values can also be specified with the functions starpu_set_limit_min_submitted_tasks() and starpu_set_limit_max_submitted_ta

An idea of how much memory is used for tasks and data handles can be obtained by setting the environment variable STARPU_MAX_MEMORY_USE to `1`.

25.11 How To Reuse Memory

When your application needs to allocate more data than the available amount of memory usable by StarPU (given by starpu_memory_get_available()), the allocation cache system can reuse data buffers used by previously executed tasks. For this system to work with MPI tasks, you need to submit tasks progressively instead of as soon as possible, because in the case of MPI receives, the allocation cache check for reusing data buffers will be done at submission time, not at execution time.

There are two options to control the task submission flow. The first one is by controlling the number of submitted tasks during the whole execution. This can be done whether by setting the environment vari-

ables STARPU_LIMIT_MAX_SUBMITTED_TASKS and STARPU_LIMIT_MIN_SUBMITTED_TASKS to tell StarPU when to stop submitting tasks and when to wake up and submit tasks again, or by explicitly calling starpu_task_wait_for_n_submitted() in your application code for finest grain control (for example, between two iterations of a submission loop).
The second option is to control the memory size of the allocation cache. This can be done in the application by using jointly starpu_memory_get_available() and starpu_memory_wait_available() to submit tasks only when there is enough memory space to allocate the data needed by the task, i.e. when enough data are available for reuse in the allocation cache.

25.12 Performance Model Calibration

Most schedulers are based on an estimation of codelet duration on each kind of processing unit. For this to be possible, the application programmer needs to configure a performance model for the codelets of the application (see Performance Model Example for instance). History-based performance models use on-line calibration. When using a scheduler which requires such performance model, StarPU will automatically calibrate codelets which have never been calibrated yet, and save the result in `$STARPU_HOME/.starpu/sampling/codelets`. The models are indexed by machine name. They can then be displayed various ways, see Performance Of Codelets .
By default, StarPU stores separate performance models according to the hostname of the system. To avoid having to calibrate performance models for each node of a homogeneous cluster for instance, the model can be shared by using `export STARPU_HOSTNAME=some_global_name` (STARPU_HOSTNAME), where `some_global_name` is the name of the cluster for instance, which thus overrides the hostname of the system.
By default, StarPU stores separate performance models for each GPU. To avoid having to calibrate performance models for each GPU of a homogeneous set of GPU devices for instance, the model can be shared by using the environment variables STARPU_PERF_MODEL_HOMOGENEOUS_CUDA, STARPU_PERF_MODEL_HOMOGENEOUS_OPENCL and STARPU_PERF_MODEL_HOMOGENEOUS_MPI_MS depending on your GPU device type.

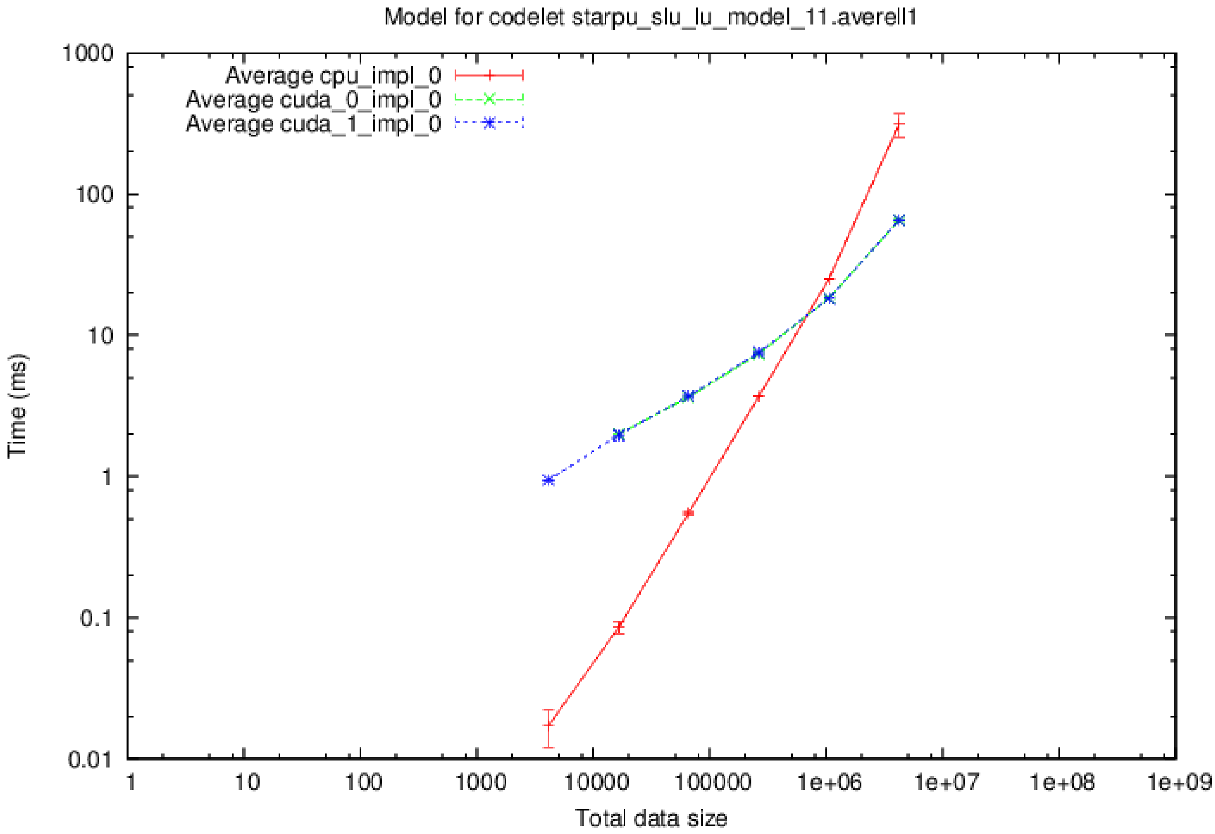
```
export STARPU_PERF_MODEL_HOMOGENEOUS_CUDA=1
export STARPU_PERF_MODEL_HOMOGENEOUS_OPENCL=1
export STARPU_PERF_MODEL_HOMOGENEOUS_MPI_MS=1
```

To force continuing calibration, use `export STARPU_CALIBRATE=1` (STARPU_CALIBRATE). This may be necessary if your application has not-so-stable performance. It may also be useful to use `STARPU_SCHED=eager` to get tasks distributed over the various workers. StarPU will force calibration (and thus ignore the current result) until 10 (`_STARPU_CALIBRATION_MINIMUM`) measurements have been made on each architecture, to avoid bad scheduling decisions just because the first measurements were not so good.
Note that StarPU will not record the very first measurement for a given codelet and a given size, because it would most often be hit by computation library loading or initialization. StarPU will also throw measurements away if it notices that after computing an average execution time, it notices that most subsequent tasks have an execution time largely outside the computed average ("Too big deviation for model..." warning messages). By looking at the details of the message and their reported measurements, it can highlight that your computation library really has non-stable measurements, which is probably an indication of an issue in the computation library, or the execution environment (e.g. rogue daemons).
Details on the current performance model status can be obtained with the tool `starpu_perfmodel_display`: the option `-l` lists the available performance models, and the option `-s` allows choosing the performance model to be displayed. The result looks like:

```
$ starpu_perfmodel_display -s starpu_slu_lu_model_getrf
performance model for cpu_impl_0
# hash      size      flops         mean          dev           n
914f3bef   1048576   0.000000e+00  2.503577e+04  1.982465e+02  8
3e921964   65536     0.000000e+00  5.527003e+02  1.848114e+01  7
e5a07e31   4096      0.000000e+00  1.717457e+01  5.190038e+00  14
...
```

It shows that for the LU 11 kernel with a 1MiB matrix, the average execution time on CPUs was about 25ms, with a 0.2ms standard deviation, over 8 samples. It is a good idea to check this before doing actual performance measurements.
A graph can be drawn by using the tool `starpu_perfmodel_plot`:

```
$ starpu_perfmodel_plot -s starpu_slu_lu_model_getrf
4096 16384 65536 262144 1048576 4194304
$ gnuplot starpu_starpu_slu_lu_model_getrf.gp
$ gv starpu_starpu_slu_lu_model_getrf.eps
```



If a kernel source code was modified (e.g. performance improvement), the calibration information is stale and should be dropped, to re-calibrate from start. This can be done by using `export STARPU_CALIBRATE=2` (STARPU_CALIBRATE).

Note: history-based performance models get calibrated only if a performance-model-based scheduler is chosen.

The history-based performance models can also be explicitly filled by the application without execution, if e.g. the application already has a series of measurements. This can be done by using starpu_perfmodel_update_history(), for instance:

```
static struct starpu_perfmodel perf_model =
{
    .type = STARPU_HISTORY_BASED,
    .symbol = "my_perfmodel",
};
struct starpu_codelet cl =
{
    .cuda_funcs = { cuda_func1, cuda_func2 },
    .nbuffers = 1,
    .modes = {STARPU_W},
    .model = &perf_model
};
void feed(void)
{
    struct my_measure *measure;
    struct starpu_task task;
    starpu_task_init(&task);
    task.cl = &cl;
    for (measure = &measures[0]; measure < measures[last]; measure++)
    {
        starpu_data_handle_t handle;
        starpu_vector_data_register(&handle, -1, 0, measure->size, sizeof(float));
        task.handles[0] = handle;
        starpu_perfmodel_update_history(&perf_model, &task, STARPU_CUDA_DEFAULT + measure->cudadev, 0,
      measure->implementation, measure->time);
        starpu_task_clean(&task);
        starpu_data_unregister(handle);
    }
}
```

Measurement has to be provided in milliseconds for the completion time models, and in Joules for the energy consumption models.

25.13 Profiling

A quick view of how many tasks each worker has executed can be obtained by setting `export STARPU_↩ WORKER_STATS=1` (STARPU_WORKER_STATS). This is a convenient way to check that execution did happen on accelerators, without penalizing performance with the profiling overhead. The environment variable STARPU_WORKER_STATS_FILE can be defined to specify a filename in which to display statistics, by default statistics are printed on the standard error stream.
A quick view of how much data transfers have been issued can be obtained by setting `export STARPU_BUS↩ _STATS=1` (STARPU_BUS_STATS). The environment variable STARPU_BUS_STATS_FILE can be defined to specify a filename in which to display statistics, by default statistics are printed on the standard error stream.
More detailed profiling information can be enabled by using `export STARPU_PROFILING=1` (STARPU_PROFILING) or by calling starpu_profiling_status_set() from the source code. Statistics on the execution can then be obtained by using `export STARPU_BUS_STATS=1` and `export STARPU_WORKER_STATS=1` . More details on performance feedback are provided in the next chapter.

25.14 Overhead Profiling

Offline Performance Tools can already provide an idea of to what extent and which part of StarPU brings an overhead on the execution time. To get a more precise analysis of which parts of StarPU bring the most overhead, `gprof` can be used.
First, recompile and reinstall StarPU with `gprof` support:

```
../configure --enable-perf-debug --disable-shared --disable-build-tests --disable-build-examples
```

Make sure not to leave a dynamic version of StarPU in the target path: remove any remaining `libstarpu-*.so`
Then relink your application with the static StarPU library, make sure that running `ldd` on your application does not mention any `libstarpu` (i.e. it's really statically-linked).

```
gcc test.c -o test $(pkg-config --cflags starpu-1.4) $(pkg-config --libs starpu-1.4)
```

Now you can run your application, this will create a file `gmon.out` in the current directory, it can be processed by running `gprof` on your application:

```
gprof ./test
```

This will dump an analysis of the time spent in StarPU functions.

Chapter 26

Frequently Asked Questions

26.1 How To Initialize A Computation Library Once For Each Worker?

Some libraries need to be initialized once for each concurrent instance that may run on the machine. For instance, a C++ computation class which is not thread-safe by itself, but for which several instantiated objects of that class can be used concurrently. This can be used in StarPU by initializing one such object per worker. For instance, the `libstarpufft` example does the following to be able to use FFTW on CPUs.

Some global array stores the instantiated objects:

```
fftw_plan plan_cpu[STARPU_NMAXWORKERS];
```

At initialization time of libstarpu, the objects are initialized:

```
int workerid;
for (workerid = 0; workerid < starpu_worker_get_count(); workerid++)
{
    switch (starpu_worker_get_type(workerid))
    {
        case STARPU_CPU_WORKER:
            plan_cpu[workerid] = fftw_plan(...);
            break;
    }
}
```

And in the codelet body, they are used:

```
static void fft(void *descr[], void *_args)
{
    int workerid = starpu_worker_get_id();
    fftw_plan plan = plan_cpu[workerid];
    ...
    fftw_execute(plan, ...);
}
```

We call starpu_worker_get_id() to retrieve the worker ID associated with the currently executing task, or call starpu_worker_get_id_check() with the error checking.

This however is not sufficient for FFT on CUDA: initialization has to be done from the workers themselves. This can be done thanks to starpu_execute_on_each_worker() or starpu_execute_on_each_worker_ex() with a specified task name, or starpu_execute_on_specific_workers() with specified workers. For instance, `libstarpufft` does the following.

```
static void fft_plan_gpu(void *args)
{
    plan plan = args;
    int n2 = plan->n2[0];
    int workerid = starpu_worker_get_id();
    cufftPlan1d(&plan->plans[workerid].plan_cuda, n, _CUFFT_C2C, 1);
    cufftSetStream(plan->plans[workerid].plan_cuda, starpu_cuda_get_local_stream());
}
void starpufft_plan(void)
{
    starpu_execute_on_each_worker(fft_plan_gpu, plan, STARPU_CUDA);
}
```

26.2 Hardware Topology

26.2.1 Interoperability hwloc

If hwloc is used, we can call starpu_get_hwloc_topology() to get the hwloc topology used by StarPU, and call starpu_get_pu_os_index() to get the OS index of a PU. We can call starpu_worker_get_hwloc_cpuset() to retrieve the hwloc CPU set associated with a worker.

26.2.2 Memory

There are various functions that we can use to retrieve information of memory node, such as to get the name of a memory node we call starpu_memory_node_get_name() and to get the kind of a memory node we call starpu_node_get_kind(). To retrieve the device ID associated with a memory node we call starpu_memory_node_get_devid(). We can call starpu_worker_get_local_memory_node() to retrieve the local memory node associated with the current worker. We can also specify a worker and call starpu_worker_get_memory_node() to retrieve the associated memory node. To get the type of memory node associated with a kind of worker we call starpu_worker_get_memory_node_kind(). If we want to know the total number of memory nodes in the system we can call starpu_memory_nodes_get_count(), and we can also retrieve the total number of memory nodes in the system that match a specific memory node kind by calling starpu_memory_nodes_get_count_by_kind(). We can call starpu_memory_node_get_ids_by_type() to get the identifiers of memory nodes in the system that match a specific memory node type. To obtain a bitmap representing logical indexes of NUMA nodes we can call starpu_get_memory_location_bitmap().

26.2.3 Workers

StarPU provides a range of functions for querying and managing the worker configurations on a given system. One such function is starpu_worker_get_count(), which returns the total number of workers in the system. In addition to this, there are also specific functions to obtain the number of workers associated with various processing units controlled by StarPU: to retrieve the number of CPUs we can call starpu_cpu_worker_get_count(), to retrieve the number of CUDA devices we can call starpu_cuda_worker_get_count(), to retrieve the number of HIP devices we can call starpu_hip_worker_get_count(), to retrieve the number of OpenCL devices we can call starpu_opencl_worker_get_count(), to retrieve the number of MPI Master Slave workers we can call starpu_mpi_ms_worker_get_count(), and to retrieve the number of TCPIP Master Slave workers we can call starpu_tcpip_ms_worker_get_count().

There are various functions that we can use to retrieve information of the worker. We call starpu_worker_get_name() to get the name of the worker, we call starpu_worker_get_devid() to get the device ID of the worker or call starpu_worker_get_devids() to retrieve the list of device IDs that are associated with a worker, and call starpu_worker_get_devnum() to get number of the device controlled by the worker which begin from 0. We call starpu_worker_get_subworkerid() to get the ID of sub-worker for the device. We call starpu_worker_get_sched_ctx_list() to retrieve a list of scheduling contexts that a worker is associated with. We call starpu_worker_get_stream_workerids() to retrieve the list of worker IDs that share the same stream as a given worker.

To retrieve the total number of NUMA nodes in the system we call starpu_memory_nodes_get_numa_count(). To get the device identifier associated with a specific NUMA node and to get the NUMA node identifier associated with a specific device we can call starpu_memory_nodes_numa_id_to_devid() and starpu_memory_nodes_numa_devid_to_id() respectively.

We can also print out information about the workers currently registered with StarPU. starpu_worker_display_all() prints out information of all workers, starpu_worker_display_names() prints out information of all the workers of the given type, starpu_worker_display_count() prints out the number of workers of the given type.

StarPU provides various functions associated to the type of processing unit, such as starpu_worker_get_type(), which returns the type of processing unit associated to the worker, e.g. CPU or CUDA. We can call starpu_worker_get_type_as_string() to retrieve a string representation of the type of a worker or call starpu_worker_get_type_from_string() to retrieve a worker type enumeration value from a string representation of a worker type or call starpu_worker_get_type_as_env_var() to retrieve a string representation of the type of a worker that can be used as an environment variable. Another function, starpu_worker_get_count_by_type(), returns the number of workers of a specific type. starpu_worker_get_ids_by_type() returns a list of worker IDs for a specific type, and starpu_worker_get_by_type() returns the ID of the specific worker that has the specific type, starpu_worker_get_by_devid() returns the ID of the worker that has the specific type and device ID. To get the type of worker associated with a kind of memory node we call starpu_memory_node_get_worker_archtype(). To check if type of processing unit matches one of StarPU's defined worker architectures we can call starpu_worker_archtype_is_valid(), while in order to convert an architecture mask to a worker architecture we can call starpu_arch_mask_to_worker_archtype().

To retrieve the binding ID of the worker associated with the currently executing task we can call starpu_worker_get_bindid(), it is useful for applications that require information about the binding of a particular task to a specific processor. We can call starpu_bindid_get_workerids() to retrieve the list of worker IDs that are bound to a given binding ID.

We can call starpu_workers_get_tree() to get information about the tree facilities provided by StarPU.

26.2.4 Bus

StarPU provides several functions to declare or retrieve information about the buses in a machine. The function starpu_bus_get_count() can be used to get the total number of buses available. To obtain the identifier of the bus between a source and destination point, the function starpu_bus_get_id() can be called. The source and destination points of a bus can be obtained by calling the functions starpu_bus_get_src() and starpu_bus_get_dst() respectively. Furthermore, users can use the function starpu_bus_set_direct() to declare that there is a direct link between a GPU and memory to the driver. The direct link can significantly reduce data transfer latency and improve overall performance. Moreover, users can use the function starpu_bus_get_direct() to retrieve information about whether a direct link has been established between a GPU and memory using the starpu_bus_set_direct() function. starpu_bus_set_ngpus() and starpu_bus_get_ngpus() functions can be used to declare and retrieve the number of GPUs of this bus that users need.

26.3 Using The Driver API

Running Drivers

```
int ret;
struct starpu_driver =
{
    .type = STARPU_CUDA_WORKER,
    .id.cuda_id = 0
};
ret = starpu_driver_init(&d);
if (ret != 0)
    error();
while (some_condition)
{
    ret = starpu_driver_run_once(&d);
    if (ret != 0)
        error();
}
ret = starpu_driver_deinit(&d);
if (ret != 0)
    error();
```

same as:

```
int ret;
struct starpu_driver =
{
    .type = STARPU_CUDA_WORKER,
    .id.cuda_id = 0
};
ret = starpu_driver_run(&d);
if (ret != 0)
    error();
```

The function starpu_driver_run() initializes the given driver, run it until starpu_drivers_request_termination() is called.

To add a new kind of device to the structure starpu_driver, one needs to:

1. Add a member to the union starpu_driver::id
2. Modify the internal function `_starpu_launch_drivers()` to make sure the driver is not always launched.
3. Modify the function starpu_driver_run() so that it can handle another kind of architecture. The function starpu_driver_run() is equal to call starpu_driver_init(), then to call starpu_driver_run_once() in a loop, and finally to call starpu_driver_deinit().
4. Write the new function `_starpu_run_foobar()` in the corresponding driver.

26.4 On-GPU Rendering

Graphical-oriented applications need to draw the result of their computations, typically on the very GPU where these happened. Technologies such as OpenGL/CUDA interoperability permit to let CUDA directly work on the OpenGL buffers, making them thus immediately ready for drawing, by mapping OpenGL buffer, textures or renderbuffer objects into CUDA. CUDA however imposes some technical constraints: peer memcpy has to be disabled, and the thread that runs OpenGL has to be the one that runs CUDA computations for that GPU.

To achieve this with StarPU, pass the option --disable-cuda-memcpy-peer to `configure` (TODO: make it dynamic), OpenGL/GLUT has to be initialized first, and the interoperability mode has to be enabled by using the field starpu_conf::cuda_opengl_interoperability, and the driver loop has to be run by the application, by using the field starpu_conf::not_launched_drivers to prevent StarPU from running it in a separate thread, and by using starpu_driver_run() to run the loop. The examples `gl_interop` and `gl_interop_idle` show how it articulates in a simple case, where rendering is done in task callbacks. The former uses `glutMainLoopEvent` to make GLUT progress from the StarPU driver loop, while the latter uses `glutIdleFunc` to make StarPU progress from the GLUT main loop.

Then, to use an OpenGL buffer as a CUDA data, StarPU simply needs to be given the CUDA pointer at registration, for instance:

```
/* Get the CUDA worker id */
for (workerid = 0; workerid < starpu_worker_get_count(); workerid++)
        if (starpu_worker_get_type(workerid) == STARPU_CUDA_WORKER)
                break;
/* Build a CUDA pointer pointing at the OpenGL buffer */
cudaGraphicsResourceGetMappedPointer((void**)&output, &num_bytes, resource);
/* And register it to StarPU */
starpu_vector_data_register(&handle, starpu_worker_get_memory_node(workerid), output, num_bytes /
      sizeof(float4), sizeof(float4));
/* The handle can now be used as usual */
starpu_task_insert(&cl, STARPU_RW, handle, 0);
/* ...  */
/* This gets back data into the OpenGL buffer */
starpu_data_unregister(handle);
```

and display it e.g. in the callback function.

26.5 Using StarPU With MKL 11 (Intel Composer XE 2013)

Some users had issues with MKL 11 and StarPU (versions 1.1rc1 and 1.0.5) on Linux with MKL, using 1 thread for MKL and doing all the parallelism using StarPU (no multithreaded tasks), setting the environment variable `MKL_NUM_THREADS` to `1`, and using the threaded MKL library, with `iomp5`.

Using this configuration, StarPU only uses 1 core, no matter the value of STARPU_NCPU. The problem is actually a thread pinning issue with MKL.

The solution is to set the environment variable KMP_AFFINITY to `disabled` (http://software.intel.com/sites/products/documentation/studio/composer/en-us/2011Update/compiler_c/optaps/common/optaps_openmp_thread_affinity.htm).

26.6 Thread Binding on NetBSD

When using StarPU on a NetBSD machine, if the topology discovery library `hwloc` is used, thread binding will fail. To prevent the problem, you should at least use the version 1.7 of `hwloc`, and also issue the following call:

```
$ sysctl -w security.models.extensions.user_set_cpu_affinity=1
```

Or add the following line in the file `/etc/sysctl.conf`

```
security.models.extensions.user_set_cpu_affinity=1
```

26.7 StarPU permanently eats 100% of all CPUs

Yes, this is on purpose.

By default, StarPU uses active polling on task queues to minimize wake-up latency for better overall performance.

We can call starpu_is_paused() to check whether the task processing by workers has been paused or not.

If eating CPU time is a problem (e.g. application running on a desktop), pass option --enable-blocking-drivers to `configure`. This will add some overhead when putting CPU workers to sleep or waking them, but avoid eating 100% CPU permanently.

26.8 Interleaving StarPU and non-StarPU code

If your application only partially uses StarPU, and you do not want to call starpu_init() / starpu_shutdown() at the beginning/end of each section, StarPU workers will poll for work between the sections. To avoid this behavior, you can "pause" StarPU with the starpu_pause() function. This will prevent the StarPU workers from accepting new work (tasks that are already in progress will not be frozen), and stop them from polling for more work.

Note that this does not prevent you from submitting new tasks, but they won't execute until starpu_resume() is called. Also note that StarPU must not be paused when you call starpu_shutdown(), and that this function pair works in a push/pull manner, i.e. you need to match the number of calls to these functions to clear their effect.

One way to use these functions could be:

```
starpu_init(NULL);
starpu_worker_wait_for_initialisation(); // Wait for the worker to complete its initialization process
starpu_pause(); // To submit all the tasks without a single one executing
submit_some_tasks();
starpu_resume(); // The tasks start executing
starpu_task_wait_for_all();
starpu_pause(); // Stop the workers from polling
starpu_resume();
starpu_shutdown();
```

26.9 When running with CUDA or OpenCL devices, I am seeing less CPU cores

Yes, this is on purpose.

Since GPU devices are way faster than CPUs, StarPU needs to react quickly when a task is finished, to feed the GPU with another task (StarPU actually submits a couple of tasks in advance to pipeline this, but filling the pipeline still has to be happening often enough), and thus it has to dedicate threads for this, and this is a very CPU-consuming duty. StarPU thus dedicates one CPU core for driving each GPU by default.

Such dedication is also useful when a codelet is hybrid, i.e. while kernels are running on the GPU, the codelet can run some computation, which thus be run by the CPU core instead of driving the GPU.

One can choose to dedicate only one thread for all the CUDA devices by setting the STARPU_CUDA_THREAD_PER_DEV environment variable to `1`. The application however should use STARPU_CUDA_ASYNC on its CUDA codelets (asynchronous execution), otherwise the execution of a synchronous CUDA codelet will monopolize the thread, and other CUDA devices will thus starve while it is executing.

26.10 StarPU does not see my CUDA device

First, make sure that CUDA is properly running outside StarPU: build and run the following program with `-lcudart` :

```
#include <stdio.h>
#include <cuda.h>
#include <cuda_runtime.h>
int main(void)
{
        int n, i, version;
        cudaError_t err;
        err = cudaGetDeviceCount(&n);
        if (err)
        {
                fprintf(stderr,"cuda error %d\n", err);
                exit(1);
        }
        cudaDriverGetVersion(&version);
        printf("driver version %d\n", version);
        cudaRuntimeGetVersion(&version);
        printf("runtime version %d\n", version);
        printf("\n");
        for (i = 0; i < n; i++)
        {
                struct cudaDeviceProp props;
                printf("CUDA%d\n", i);
                err = cudaGetDeviceProperties(&props, i);
                if (err)
                {
                        fprintf(stderr,"cudaGetDeviceProperties cuda error %d\n", err);
                        continue;
                }
                printf("%s\n", props.name);
                printf("%0.3f GB\n", (float) props.totalGlobalMem / (1«30));
                printf("%u MP\n", props.multiProcessorCount);
                printf("\n");
                err = cudaSetDevice(i);
                if (err)
                {
                        fprintf(stderr,"cudaSetDevice(%d) cuda error %d\n", err, i);
                        continue;
                }
                err = cudaFree(0);
                if (err)
                {
                        fprintf(stderr,"cudaFree(0) on %d cuda error %d\n", err, i);
                        continue;
                }
        }
        return 0;
}
```

If that program does not find your device, the problem is not at the StarPU level, but with the CUDA drivers, check the documentation of your CUDA setup. This program is available in the source directory of StarPU in `tools/gpus/check_cuda.c`, along with another CUDA program `tools/gpus/cuda_list.cu`.

26.11 StarPU does not see my OpenCL device

First, make sure that OpenCL is properly running outside StarPU: build and run the following program with `-lOpenCL`:

```
#include <CL/cl.h>
#include <stdio.h>
#include <assert.h>
int main(void)
{
    cl_device_id did[16];
    cl_int err;
    cl_platform_id pid, pids[16];
    cl_uint nbplat, nb;
    char buf[128];
    size_t size;
    int i, j;
    err = clGetPlatformIDs(sizeof(pids)/sizeof(pids[0]), pids, &nbplat);
    assert(err == CL_SUCCESS);
    printf("%u platforms\n", nbplat);
    for (j = 0; j < nbplat; j++)
    {
        pid = pids[j];
        printf("    platform %d\n", j);
        err = clGetPlatformInfo(pid, CL_PLATFORM_VERSION, sizeof(buf)-1, buf, &size);
        assert(err == CL_SUCCESS);
        buf[size] = 0;
        printf("        platform version %s\n", buf);
        err = clGetDeviceIDs(pid, CL_DEVICE_TYPE_ALL, sizeof(did)/sizeof(did[0]), did, &nb);
        if (err == CL_DEVICE_NOT_FOUND)
          nb = 0;
        else
          assert(err == CL_SUCCESS);
        printf("%d devices\n", nb);
        for (i = 0; i < nb; i++)
        {
            err = clGetDeviceInfo(did[i], CL_DEVICE_VERSION, sizeof(buf)-1, buf, &size);
            buf[size] = 0;
            printf("    device %d version %s\n", i, buf);
        }
    }
    return 0;
}
```

If that program does not find your device, the problem is not at the StarPU level, but with the OpenCL drivers, check the documentation of your OpenCL implementation. This program is available in the source directory of StarPU in `tools/gpus/check_opencl.c`.

26.12 There seems to be errors when copying to and from CUDA devices

You should first try to disable asynchronous copies between CUDA and CPU workers. You can either do that with the configuration parameter --disable-asynchronous-cuda-copy or with the environment variable STARPU_DISABLE_ASYNCHRONOUS_CUDA_COPY.

If your application keeps failing, you will find in the source directory of StarPU, a directory named `tools/gpus` with various programs. `cuda_copy.cu` is testing the direct or undirect copy between CUDA devices.

You can also try to just disable the direct gpu-gpu transfers (known to fail under some hardware/cuda combinations) by setting the STARPU_ENABLE_CUDA_GPU_GPU_DIRECT environment variable to 0.

26.13 I keep getting a "Incorrect performance model file" error

The performance model file, used by StarPU to record the performance of codelets, seem to have been corrupted. Perhaps a previous run of StarPU stopped abruptly, and thus could not save it properly. You can have a look at the file if you can fix it, but the simplest way is to just remove the file and run again, StarPU will just have to re-perform calibration for the corresponding codelet.

Part VI

StarPU Language Bindings

Chapter 27

Organization

This part shows how StarPU which is natively written in C, has been extended to allow applications written in other languages to use it.

- You can learn to natively access most of StarPU functionalities from Fortran 2008+ codes with some explanations and examples in Chapter The StarPU Native Fortran Support.
- You can find out how to execute Java applications with some important StarPU APIs in Chapter StarPU Java Interface.
- Python interface supports most of the main StarPU functionalities, and new functions especially adapted to Python have been added as well. There are detailed explanations and examples in Chapter Python Interface.
- You can learn how to execute OpenMP tasks with some specific functions in Chapter The StarPU OpenMP Runtime Support (SO

Chapter 28

Native Fortran Support

StarPU provides the necessary routines and support to natively access most of its functionalities from Fortran 2008+ codes.
All symbols (functions, constants) are defined in `fstarpu_mod.f90`. Every symbol of the Native Fortran support API is prefixed by `fstarpu_`.
Note: Mixing uses of `fstarpu_` and `starpu_` symbols in the same Fortran code has unspecified behavior. See Valid API Mixes and Language Mixes for a discussion about valid and unspecified combinations.

28.1 Implementation Details and Specificities

28.1.1 Prerequisites

The Native Fortran support relies on Fortran 2008 specific constructs, as well as on the support for interoperability of assumed-shape arrays introduced as part of Fortran's Technical Specification ISO/IEC TS 29113:2012, for which no equivalent are available in previous versions of the standard. It has currently been tested successfully with GNU GFortran 4.9, GFortran 5.x, GFortran 6.x and the Intel Fortran Compiler $>=$ 2016. It is known not to work with GNU GFortran $<$ 4.9, Intel Fortran Compiler $<$ 2016.
See Section Using StarPU with Older Fortran Compilers for information on how to write StarPU Fortran code with older compilers.

28.1.2 Configuration

The Native Fortran API is enabled and its companion `fstarpu_mod.f90` Fortran module source file is installed by default when a Fortran compiler is found, unless the detected Fortran compiler is known not to support the requirements for the Native Fortran API. The support can be disabled through the `configure` option --disable-fortran. Conditional compiled source codes may check for the availability of the Native Fortran Support by testing whether the preprocessor macro `STARPU_HAVE_FC` is defined or not.

28.1.3 Examples

Several examples using the Native Fortran API are provided in StarPU's `examples/native_fortran/` examples directory, to showcase the Fortran flavor of various basic and more advanced StarPU features.

28.1.4 Compiling a Native Fortran Application

The Fortran module `fstarpu_mod.f90` installed in StarPU's `include/` directory provides all the necessary API definitions. It must be compiled with the same compiler (same vendor, same version) as the application itself, and the resulting `fstarpu_mod.o` object file must be linked with the application executable.
Each example provided in StarPU's `examples/native_fortran/` examples directory comes with its own dedicated Makefile for out-of-tree build. Such example Makefiles may be used as starting points for building application codes with StarPU.

28.2 Fortran Translation for Common StarPU API Idioms

All these examples assume that the standard Fortran module `iso_c_binding` is in use.

- Specifying a `NULL` pointer

  ```
  type(c_ptr) ::  my_ptr  !  variable to store the pointer
  !  [...]
  my_ptr = c_null_ptr    !  assign standard constant for null ptr
  ```

- Obtaining a pointer to some object:

  ```
  real(8), dimension(:), allocatable, target ::  va
  type(c_ptr) ::  p_va  !  variable to store a pointer to array va
  !  [...]
  p_va = c_loc(va)
  ```

- Obtaining a pointer to some subroutine:

  ```
  !  pointed routine definition
  recursive subroutine myfunc () bind(C)
  !  [...]
  type(c_funptr) ::  p_fun  !  variable to store the routine pointer
  !  [...]
  p_fun = c_funloc(my_func)
  ```

- Obtaining the size of some object:

  ```
  real(8) ::  a
  integer(c_size_t) ::  sz_a  !  variable to store the size of a
  !  [...]
  sz_a = c_sizeof(a)
  ```

- Obtaining the length of an array dimension:

  ```
  real(8), dimension(:,:), allocatable, target ::  vb
  integer(c_int) ::  ln_vb_1  !  variable to store the length of vb's dimension 1
  integer(c_int) ::  ln_vb_2  !  variable to store the length of vb's dimension 2
  !  [...]
  ln_vb_1 = 1+ubound(vb,1)-lbound(vb,1)  !  get length of dimension 1 of vb
  ln_vb_2 = 1+ubound(vb,2)-lbound(vb,2)  !  get length of dimension 2 of vb
  ```

- Specifying a string constant:

  ```
  type(c_ptr) ::  my_cl  !  a StarPU codelet
  !  [...]
  !  set the name of a codelet to string 'my_codele't:
  call fstarpu_codelet_set_name(my_cl, c_char_"my_codelet"//c_null_char)
  !  note:  using the C_CHAR_ prefix and the //C_NULL_CHAR concatenation at the end ensures
  !  that the string constant is properly '\0' terminated, and compatible with StarPU's
  !  internal C routines
  !
  !  note:  plain Fortran string constants are not '\0' terminated, and as such, must not be
  !  passed to starpu routines.
  ```

- Combining multiple flag constants with a bitwise 'or':

  ```
  type(c_ptr) ::  my_cl  !  a pointer for the codelet structure
  !  [...]
  !  add a managed buffer to a codelet, specifying both the Read/Write access mode and the Locality hint
  call fstarpu_codelet_add_buffer(my_cl, fstarpu_rw.ior.fstarpu_locality)
  ```

A basic example is available in `examples/native_fortran/nf_vector_scal.f90`.

28.3 Uses, Initialization and Shutdown

The snippet below show an example of minimal StarPU code using the Native Fortran support. The program should `use` the standard module `iso_c_binding` as well as StarPU's `fstarpu_mod`. The StarPU runtime engine is initialized with a call to function `fstarpu_init`, which returns an integer status of 0 if successful or non-0 otherwise. Eventually, a call to `fstarpu_shutdown` ends the runtime engine and frees all internal StarPU data structures.

```
program nf_initexit
        use iso_c_binding       !  C interfacing module
        use fstarpu_mod         !  StarPU interfacing module
        implicit none           !  Fortran recommended best practice
        integer(c_int) ::  err   !  return status for fstarpu_init
        !  initialize StarPU with default settings
        err = fstarpu_init(c_null_ptr)
        if (err /= 0) then
                stop 1          !  StarPU initialization failure
end if
        !  - add StarPU Native Fortran API calls here
        !  shut StarPU down
        call fstarpu_shutdown()
end program nf_initexit
```

28.4 Fortran Flavor of StarPU's Variadic Insert_task

Fortran does not have a construction similar to C variadic functions, on which starpu_task_insert() relies at the time of this writing. However, Fortran's variable length arrays of `c_ptr` elements enable to emulate much of the convenience of C's variadic functions. This is the approach retained for implementing `fstarpu_task_insert`.

The general syntax for using `fstarpu_task_insert` is as follows:

```
call fstarpu_task_insert((/ <codelet ptr>       &
    [, <access mode flags>, <data handle>]*     &
    [, <argument type constant>, <argument>]*   &
    , c_null_ptr /))
```

There is thus a unique array argument `(/ ... /)` passed to `fstarpu_task_insert` which itself contains the task settings. Each element of the array must be of type `type(c_ptr)`. The last element of the array must be `C_NULL_PTR`.

Example extracted from nf_vector.f90:

```
call fstarpu_task_insert((/ cl_vec,          &    !  codelet
    fstarpu_r, dh_va,                        &    !  a first data handle
    fstarpu_rw.ior.fstarpu_locality, dh_vb,  &    !  a second data handle
    c_null_ptr /))                                !  no more args
```

The full example is available in `examples/native_fortran/nf_vector.f90`.

28.5 Functions and Subroutines Expecting Data Structures Arguments

Several StarPU structures that are expected to be passed to the C API, are replaced by function/subroutine wrapper sets to allocate, set fields and free such structure. This strategy has been preferred over defining native Fortran equivalent of such structures using Fortran's derived types, to avoid potential layout mismatch between C and Fortran StarPU data structures. Examples of such data structures wrappers include `fstarpu_conf_allocate` and alike, `fstarpu_codelet_allocate` and alike, `fstarpu_data_filter_allocate` and alike.

Here is an example of allocating, filling and deallocating a codelet structure:

```
!  a pointer for the codelet structure
type(c_ptr) ::  cl_vec
!  [...]
!  allocate an empty codelet structure
cl_vec = fstarpu_codelet_allocate()
!  add a CPU implementation function to the codelet
call fstarpu_codelet_add_cpu_func(cl_vec, c_funloc(cl_cpu_func_vec))
!  add a CUDA implementation function to the codelet
call fstarpu_codelet_add_cuda_func(cl_vec, c_funloc(cl_cuda_func_vec))
!  set the codelet name
call fstarpu_codelet_set_name(cl_vec, c_char_"my_vec_codelet"//c_null_char)
!  add a Read-only mode data buffer to the codelet
call fstarpu_codelet_add_buffer(cl_vec, fstarpu_r)
!  add a Read-Write mode data buffer to the codelet
call fstarpu_codelet_add_buffer(cl_vec, fstarpu_rw.ior.fstarpu_locality)
!  [...]
!  free codelet structure
call fstarpu_codelet_free(cl_vec)
```

The full example is available in `examples/native_fortran/nf_vector.f90`.

28.6 Additional Notes about the Native Fortran Support

28.6.1 Using StarPU with Older Fortran Compilers

When using older compilers, Fortran applications may still interoperate with StarPU using C marshalling functions as examplified in StarPU's `examples/fortran/` and `examples/fortran90/` example directories, though the process will be less convenient.

Basically, the main FORTRAN code calls some C wrapper functions to submit tasks to StarPU. Then, when StarPU starts a task, another C wrapper function calls the FORTRAN routine for the task.

Note that this marshalled FORTRAN support remains available even when specifying `configure` option --disable-fortran (which only disables StarPU's native Fortran layer).

28.6.2 Valid API Mixes and Language Mixes

Mixing uses of `fstarpu_` and `starpu_` symbols in the same Fortran code has unspecified behavior. Using `fstarpu_` symbols in C code has unspecified behavior.

For multi-language applications using both C and Fortran source files:

- C source files must use `starpu_` symbols exclusively
- Fortran sources must uniformly use either `fstarpu_` symbols exclusively, or `starpu_` symbols exclusively. Every other combination has unspecified behavior.

Chapter 29

StarPU Java Interface

The StarPU Java Interface provides the ability to execute Java applications on top of StarPU.

The interface allows to write either StarPU-like applications

```
package fr.labri.hpccloud.starpu.examples;
import fr.labri.hpccloud.starpu.Codelet;
import fr.labri.hpccloud.starpu.StarPU;
import fr.labri.hpccloud.starpu.data.DataHandle;
import fr.labri.hpccloud.starpu.data.IntegerVariableHandle;
import fr.labri.hpccloud.starpu.data.VectorHandle;
import java.util.Random;
import static fr.labri.hpccloud.starpu.data.DataHandle.AccessMode.*;
public class VectorScal
{
        public static final int NX = 10;
        public static final Float factor = 3.14f;
        static final Codelet scal = new Codelet()
        {
                @Override
                public void run(DataHandle[] buffers)
                {
                        VectorHandle<Float> array = (VectorHandle<Float>)buffers[0];
                        int n = array.getSize();
                        System.out.println(String.format("scaling array %s with %d elements", array, n));
                        for (int i = 0; i < n; i++)
                        {
                                array.setValueAt(i, factor * array.getValueAt(i));
                        }
                }
                @Override
                public DataHandle.AccessMode[] getAccessModes()
                {
                        return new DataHandle.AccessMode[]
                        {
                                STARPU_RW
                        };
                }
        };
        public static void main(String[] args) throws Exception
        {
                int nx = (args.length == 0) ?  NX : Integer.valueOf(args[0]);
                compute(nx);
        }
        public static void compute(int nx) throws Exception
        {
                StarPU.init();
                System.out.println(String.format("VECTOR[#nx=%d]", nx));
                VectorHandle<Float> arrayHandle = VectorHandle.register(nx);
                System.out.println(String.format("scaling array %s", arrayHandle));
                for(int i=0 ; i<nx ; i++)
                {
                        arrayHandle.setValueAt(i, i+1.0f);
                }
                StarPU.submitTask(scal, false, arrayHandle);
                arrayHandle.acquire();
                for(int i=0 ; i<nx ; i++)
                {
                        System.out.println(String.format("v[%d] = %f", i, arrayHandle.getValueAt(i)));
                }
                arrayHandle.release();
                arrayHandle.unregister();
                StarPU.shutdown();
        }
}
```

or Spark applications.

```
package fr.labri.hpccloud.starpu.examples;
```

```
import fr.labri.hpccloud.starpu.StarPU;
import fr.labri.hpccloud.starpu.data.DataPairSet;
import fr.labri.hpccloud.starpu.data.DataSet;
import fr.labri.hpccloud.starpu.data.Tuple2;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.util.Arrays;
import java.util.regex.Pattern;
public class WordCount
{
        static InputStream openFile(String filename) throws Exception
        {
                return WordCount.class.getResourceAsStream(filename);
        }
        private static final Pattern SPACE = Pattern.compile(" ");
        public static void main(String[] args ) throws Exception
        {
                InputStream input = new FileInputStream(args[0]);
                StarPU.init();
                compute(input);
                input.close();
                StarPU.shutdown();
        }
        private static void compute(InputStream input) throws Exception
        {
                DataSet<String> lines = DataSet.readFile (input, s->s).splitByBlocks(10);
                DataSet<String> words = lines.flatMap(s ->
      Arrays.asList(SPACE.split(s)).iterator()).splitByBlocks(10);
                DataPairSet<String,Integer> ones = (DataPairSet<String,Integer>)words.mapToPair(w-> new
      Tuple2<>(w,1));
                DataPairSet<String,Integer> counts = ones.reduceByKey((c1,c2)-> c1 + c2);
                for(Tuple2<String,Integer> p :  counts.collect())
                {
                        System.out.println("("+p._1()+","+p._2()+")");
                }
        }
}
```

The installation process is not yet included in the StarPU autotools mechanism. However, a file `INSTALL.org` is provided in the `starpujni` directory to explain how to proceed with the installation, and shows how to run some basic examples.

`hadoop` needs to be installed before running the installation process.

Chapter 30

Python Interface

This chapter presents the StarPU Python Interface. It provides for those used to the Python language a more concise and easy-to-use StarPU interface.
This interface supports most of the main StarPU functionalities. While not all features of the C API are replicated in the Python Interface, additional functions tailored for Python's ease of use have been incorporated.
Several examples using the Python API are provided in the directory `starpupy/examples/`.

30.1 Installation of the Python Interface

Calling `configure` will enable by default the StarPU Python Interface. You can also specify the option --enable-starpupy which will fail if some requirements are missing. For now, the only requirement is the availability of the `python3` interpreter.
The python modules `joblib` and `cloudpickle` are mandatory to run parallel codes.
The python module `numpy` is recommended, but not mandatory.

```
$ pip3 install joblib
$ pip3 install cloudpickle
$ pip3 install numpy
$  ../configure --enable-starpupy --enable-blocking-drivers --prefix=$HOME/usr/starpu
$ make
$ make install
```

You can then go to the directory in which StarPU is installed, and test the provided Python examples.

```
$ cd $HOME/usr/starpu
$ . ./bin/starpu_env
Setting StarPU environment for ...
$ cd lib/starpu/python
$ python3 starpu_py.py
Example 1:
Hello, world!
...
$
```

30.2 Python Parallelism

Python interpreters share the Global Interpreter Lock (GIL), which requires that at any time, one and only one thread has the right to execute a task. With Python versions up to 3.11, if the application is pure Python script, even with multi-interpreters, the program cannot be executed in parallel. The sharedGIL makes the multiple interpreters execution of Python actually serial rather than parallel, and the execution of Python program is single-threaded essentially.
For the pure Python script with python versions up to 3.11, the only way to achieve parallelism is to use the master-slave mechanism (Section Master Slave Support). Parallelism may be implemented with multi-interpreters in the future Python version. Details can be found in Section Multiple Interpreters. Otherwise parallelism can be achieved when external C applications are called or external APIs e.g. BLAS API is used for Numpy objects.
Starting from python version 3.12, multiple interpreters can use a separate GIL, to allow parallelism of pure python code. This can be enabled by setting STARPUPY_OWN_GIL to 1. Some corner cases are however not supported yet in python 3.12, notably the usage of futures.

30.3 Using StarPU in Python

The StarPU module should be imported in any Python code wanting to use the StarPU Python interface.

```
import starpu
```

Before using any StarPU functionality, it is necessary to call `starpu.init()`. The function `starpu.shutdown()` should be called after all StarPU functions have been called.

```
import starpu
starpu.init()
# ...
starpu.shutdown()
```

30.3.1 Submitting Tasks

One of the fundamental aspects of StarPU is the task submission. The Python Interface greatly simplifies this process, allowing for direct calls to the submission function without any extra complexities.

The Python function used for task submission follows the format: `task_submit(options)(func, *args, **kwargs)`. In this structure:

- `func` represents any Python function.
- `args` and `kwargs` denote the function's arguments.

You can also provide the function as a string.

By submitting tasks through this function, you enable StarPU to perform optimizations for your program's execution. It's recommended to submit all tasks to ensure StarPU's efficient scheduling of the underlying tasks. It's important to note that submitted tasks do not execute immediately, and you can retrieve the return value only after the task execution.

The first set of parentheses allows to specify various options. Keep in mind that each option has a default value, and even if you're not providing any options, the parentheses should be retained. The options are as follows:

- **`name` (string, default: `None`)** : Set the name of the task. This can be useful for debugging purposes.
- **`synchronous` (unsigned, default: 0)** : If this flag is set, `task_submit()` only returns when the task has been executed (or if no worker is able to process the task). Otherwise, `task_submit()` returns immediately.
- **`priority` (int, default: 0)** : Set the level of priority for the task. This is an integer value whose value must be greater than the return value of the function `starpu.sched_get_min_priority()` (for the least important tasks), and lower or equal to the return value of the function `starpu.sched_get_max_priority()` (for the most important tasks). Default priority is defined as 0 in order to allow static task initialization. Scheduling strategies that take priorities into account can use this parameter to take better scheduling decisions, but the scheduling policy may also ignore it.
- **`color` (unsigned, default: `None`)** : Set the color of the task to be used in `dag.dot`.
- **`flops` (double, default: `None`)** : Set the number of floating points operations that the task will have to achieve. This is useful for easily getting GFlops/s curves from the function `starpu.perfmodel_plot`, and for the hypervisor load balancing.
- **`perfmodel` (string, default: `None`)** : Set the name of the performance model. This name will be used as the filename where the performance model information will be saved. After the task is executed, one can call the function `starpu.perfmodel_plot()` by giving the symbol of perfmodel to view its performance curve.

30.3.2 Returning Future Object

In order to realize asynchronous frameworks, the `task_submit()` function returns a Future object. This is an extended use of StarPU provided by the Python interface. A Future represents an eventual result of an asynchronous operation. It is an awaitable object, Coroutines can await on Future objects until they either have a result or an exception set, or until they are canceled. Some basic examples are available in the script `starpupy/examples/starpu_py.py`.

This feature needs the `asyncio` module to be imported.

```
import starpu
import asyncio
```

```
starpu.init()
def add(a, b):
    return a+b
async def main():
    fut = starpu.task_submit()(add, 1, 2)
    res = await fut
    print("The result of function is", res)
asyncio.run(main())
starpu.shutdown()
```

Execution:

```
The result of function is 3
```

When using at least the version 3.8 of python, one can also use the parameter `-m asyncio` which allows to directly use `await` instead of `asyncio.run()`.

```
$ python3 -m asyncio
>>> import asyncio
```

```
import starpu
starpu.init()
def add(a, b):
        print("The result is ready!")
        return a+b
fut = starpu.task_submit()(add, 1, 2)
```

```
The result is ready!
```

```
res = await fut
res
```

```
3
```

You can also use the decorator `starpu.delayed` to wrap a function. The function can then directly be submitted to StarPU and will automatically create a Future object.

```
@starpu.delayed
def add_deco(a, b):
        print("The result is ready!")
        return a+b
fut = add_deco(1, 2)
```

```
The result is ready!
```

```
res = await fut
res
```

```
3
```

To specify options when using the decorator, just do as follows:

```
@starpu.delayed(name="add", color=2, perfmodel="add_deco")
def add_deco(a, b):
        print("The result is ready!")
        return a+b
fut = add_deco(1, 2)
```

```
The result is ready!
```

```
res = await fut
res
```

```
3
```

A Future object can also be used for the next step calculation even before being ready. The calculation will be postponed until the Future has a result.

In this example, after submitting the first task, a Future object `fut1` is created, and it is used as an argument of a second task. The second task is submitted even without having the return value of the first task.

```
import asyncio
import starpu
import time
starpu.init()
def add(a, b):
        time.sleep(10)
        print("The first result is ready!")
        return a+b
def sub(x, a):
        print("The second result is ready!")
        return x-a
fut1 = starpu.task_submit()(add, 1, 2)
fut2 = starpu.task_submit()(sub, fut1, 1)
```

```
The first result is ready!
The second result is ready!
```

```
res = await fut2
res
```

```
2
```

30.3.3 Submit Python Objects Supporting The Buffer Protocol

The Python buffer protocol is a framework in which Python objects can expose raw byte arrays to other Python objects. This can be extremely useful to efficiently store and manipulate large arrays of data. The StarPU Python Interface allows users to use such objects as task parameters.

```
import asyncio
import starpu
import time
import numpy as np
starpu.init()
def add(a,b):
        c = np.zeros(np.size(a))
        for i in range(np.size(a)):
                c[i] = a[i] + b[i]
        return c
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
fut = starpu.task_submit()(add, a, b)
res = await fut
res
```

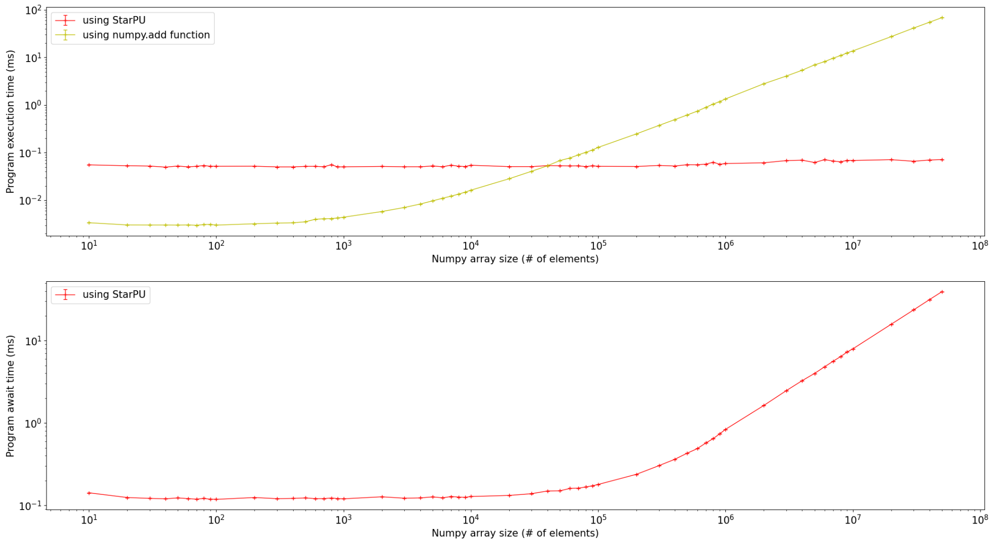
```
array([5., 7., 9.])
```

StarPU uses a specific data interface to handle Python objects supporting buffer protocol, such python objects are then managed by the StarPU data management library which allows minimizing data transfers between accelerators, and avoids copying the object each time.

We show the performances below of the `numpy` addition (`numpy.add` running the script `test_perf.sh`) with different array sizes (10, 20, ..., 100, 200, ..., 1000, 2000, ..., 10000, 20000, ..., 100000, 200000, ..., 1000000, 2000000, ..., 10000000, ..., 50000000). We compare two cases:

1. Using StarPU,
2. Without using StarPU tasks, but directly calling the `numpy.add` function.

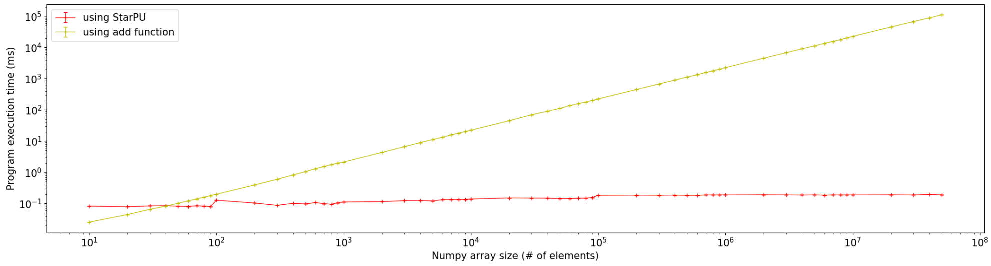
The first plot compares the task submission time when using StarPU and the program execution time without using StarPU. We can see that there is an obvious optimization using StarPU when the test array size is large. The task has not finished its execution yet as shown in second figure, the time can be used to perform other operations.

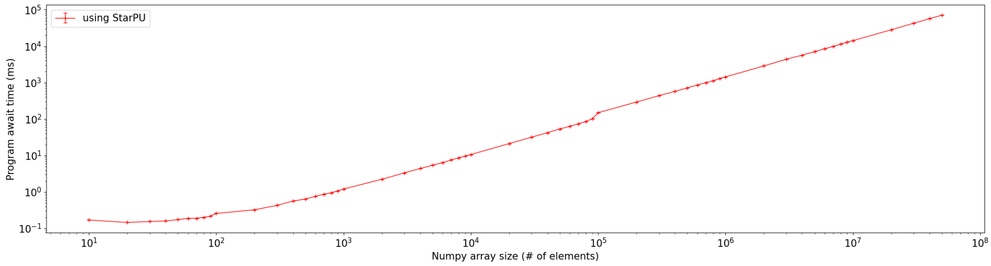


We can also define our own function to do the `numpy` operation, e.g. the element addition:

```
def add(a, b):
        for i in range(np.size(a)):
                a[i] = a[i] + b[i]
```

We will compare operation performances with the same two cases, but based on our custom function `add(a, b)`. We can see that the custom function is not as efficient as the `numpy` function overall. The optimization for large arrays is the same when using StarPU.





30.3.3.1 Access Mode Annotation

StarPU defines different access modes for a data, it can be readable (access mode is `R`), writable (access mode is `W`), or both readable and writable (access mode is `RW`). The default access mode is `R`.
For the Python interface, these modes can be defined as shown below.

1. Using the decorator `starpu.access(arg="R/W/RW")` to wrap the function.
   ```
   a = np.array([1, 2, 3, 4, 5, 6])
   e = np.array([0, 0, 0, 0, 0, 0, 0])
   @starpu.access(a="R", b="W")
   def assign(a,b):
           for i in range(min(np.size(a), np.size(b))):
                   b[i]=a[i]
   fut = starpu.task_submit()(assign, a, e)
   starpu.acquire(e)
   ```
   ```
   array([1, 2, 3, 4, 5, 6, 0])
   ```
   ```
   starpu.release(e)
   ```
2. Using the decorator `starpu.delayed(options, arg="R/W/RW")`.
   ```
   @starpu.delayed(a="R", b="W")
   def assign(a,b):
           for i in range(min(np.size(a), np.size(b))):
                   b[i]=a[i]
   fut = assign(a, e)
   starpu.acquire(e)
   ```
   ```
   array([1, 2, 3, 4, 5, 6, 0])
   ```
   ```
   starpu.release(e)
   ```
3. Using the method `starpu.set_access(func, arg="R/W/RW")` that will create a new function.
   ```
   def assign(a,b):
           for i in range(min(np.size(a), np.size(b))):
                   b[i]=a[i]
   assign_access=starpu.set_access(assign, a="R", b="W")
   fut = starpu.task_submit()(assign_access, a, e)
   starpu.acquire(e)
   ```
   ```
   array([1, 2, 3, 4, 5, 6, 0])
   ```
   ```
   starpu.release(e)
   ```

30.3.3.2 Methods

Once the access mode of one argument is set to at least `W`, it may be modified during the task execution. We should pay attention that before the task is finished, we cannot get the up-to-date value of this argument by simply using `print` function. For example:

```
import asyncio
import starpu
import time
import numpy as np
starpu.init()
a = np.array([1, 2, 3, 4, 5, 6])
e = np.array([0, 0, 0, 0, 0, 0])
@starpu.access(a="R", b="W")
def assign(a,b):
        time.sleep(10)
        for i in range(min(np.size(a), np.size(b))):
                b[i]=a[i]
fut = starpu.task_submit()(assign, a, e)
print(e) # before the task is finished
```

```
[0 0 0 0 0 0]
```

We `print` argument `e` right after submitting the task, but since the task is not finished yet, we can only get its unchanged value. If we want to get its up-to-date value, we need extra functions.
In order to access data registered to StarPU outside tasks, we provide an acquire and release mechanism.

- The `starpu.acquire(data, mode)` method should be called to access registered data outside tasks (Refer to the C API starpu_data_acquire()). StarPU will ensure that the application will get an up-to-date copy of handle in main memory located where the data was originally registered, and that all concurrent accesses (e.g. from tasks) will be consistent with the access mode specified with the given mode (`R` the default mode, `W` or `RW`).
- The `starpu.release(data)` method must be called once the application no longer needs to access the piece of data (Refer to the C API starpu_data_release()).
- The `starpu.unregister(data)` method must be called to unregister the Python object from StarPU. (Refer to the C API starpu_data_unregister()). This method waits for all calculations to be finished before unregistering data.

With `acquire`, even we ask to access the argument right after submitting the task, the up-to-date value will be printed once the task is finished.

```
starpu.acquire(e) # before the task is finished
```

```
array([1, 2, 3, 4, 5, 6, 0])
```

In order to complete the addition operation example, execution steps are:

```
import asyncio
import starpu
import time
import numpy as np
starpu.init()
@starpu.access(a="RW", b="R")
def add(a,b):
        time.sleep(10)
        for i in range(np.size(a)):
                a[i] = a[i] + b[i]
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
starpu.acquire(a, mode="R")
```

```
array([1, 2, 3])
```

```
starpu.release(a)
fut = starpu.task_submit()(add, a, b)
starpu.acquire(b, mode="R")
```

```
array([4, 5, 6])
```

```
starpu.acquire(a, mode="R") # before the task is finished
```

```
array([5, 7, 9])
```

```
starpu.release(a)
starpu.release(b)
starpu.unregister(a)
starpu.unregister(b)
```

The result of `b` is printed directly right after calling `acquire`, but the up-to-date value of `a` is printed after the task is finished. Here we need to pay attention that if we want to modify an argument during the task execution and get its up-to-date value for the future operation, we should set the access mode of this argument to at least `W`, otherwise this argument object is not synchronous, and the next task which needs this object will not wait its up-to-date value to execute.

If we call `acquire` but not `release` before the task submission, the task will not start to execute until the object is released.

An example is shown below:

```
import asyncio
import starpu
import numpy as np
import time
starpu.init()
@starpu.access(a="RW")
def add(a,b):
        print("This is the addition function")
        time.sleep(10)
        for i in range(np.size(a)):
                a[i] = a[i] + b[i]
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
starpu.acquire(a, mode="R")
```

```
array([1, 2, 3])
```

```
fut = starpu.task_submit()(add, a, b)
starpu.release(a)
```

```
This is the addition function   # The task will not start until "a" is released
```

```
starpu.acquire(a, mode="R") # Before the task is finished
```

```
array([5, 7, 9])                # After the task is finished
```

```
starpu.release(a)
starpu.unregister(a)
starpu.unregister(b)
```

30.4 StarPU Data Interface for Python Objects

StarPU uses data handles to manage a piece of data. A data handle keeps track of replicates of the same data (registered by the application) over various memory nodes. The data management library manages to keep them coherent. That also allows minimizing the data transfers, and avoids copying the object each time. Data handles are managed through specific data interfaces. Some examples applying this specific interface are available in script `starpupy/examples/starpu_py_handle.py`.

30.4.1 Interface for Ordinary Python Objects

A specific data interface has been defined to manage Python objects, such as constant (integer, float...), string, list, etc. This interface is defined with the class `Handle`. When submitting a task, instead of specifying a function and its arguments, we specify a function and the handles of its arguments.

In addition to returning a Future object, it is also possible to return a StarPU handle object when submitting a function. To do so, you need to set the `starpu.task_submit` option `ret_handle` to `True`, its default value is `False`.

```
import starpu
from starpu import Handle
starpu.init()
def add(x, y):
   return x + y
x = Handle(2)
y = Handle(3)
res = starpu.task_submit(ret_handle=True)(add, x, y)
```

We then need to call the method `get()` to get the latest version of this Python Object.

```
res.get()
```

```
5
```

When not setting the parameter `ret_handle`, the return object is a Future.

```
res_fut = starpu.task_submit()(add, x, y)
await res_fut
```

If the Python object is immutable (such as int, float, str, tuple...), registering the same object several times is authorised. That means you can do this:

```
x = Handle(2)
x1 = Handle(2)
```

x and x1 are two different Handle objects.

30.4.2 Interface for Python Objects Supporting Buffer Protocol

This StarPU data interface can also be used to manage Python objects supporting buffer protocol, i.e `numpy` array, bytes, bytearray, array.array and memoryview object.

```
import numpy as np
import starpu
from starpu import Handle
starpu.init()
def add(a,b):
   for i in range(np.size(a)):
      a[i] = a[i] + b[i]
   return a
a = np.array([1, 2, 3])
b = np.array([2, 4, 6])
a_h = Handle(a)
b_h = Handle(b)
res = starpu.task_submit(ret_handle=True)(add, a_h, b_h)
res.get()
```

```
array([3, 6, 9])
```

Different from immutable Python object, all Python objects supporting buffer protocol are mutable, and registering the same object one more time is not authorized. If you do this:

```
a = np.array([1, 2, 3])
a_h = Handle(a)
a1_h = Handle(a)
```

You will get an error message:

```
starpupy.error: Should not register the same mutable python object once more.
```

You may refer to Section Submit Python Objects Supporting The Buffer Protocol, and realize that StarPU Python interface uses data handles to manage Python objects supporting buffer protocol by default. These objects are usually relatively large, such as a big NumPy matrix. We want to avoid multiple copies and transfers of this data over various memory nodes, so we set the default `starpu.task_submit()` option `arg_handle` to `True` for users to allow their applications to get the most optimization. To deactivate the use of this data interface, you need to set the option `arg_handle` to `False`.

Since we use data handles by default, registration is implemented in the step of task submission. Therefore, you should be careful not to register again the same object after the task submission, like this:

```
a = np.array([1, 2, 3])
b = np.array([2, 4, 6])
res = starpu.task_submit(ret_handle=True)(add, a, b)
a_h = Handle(a)
```

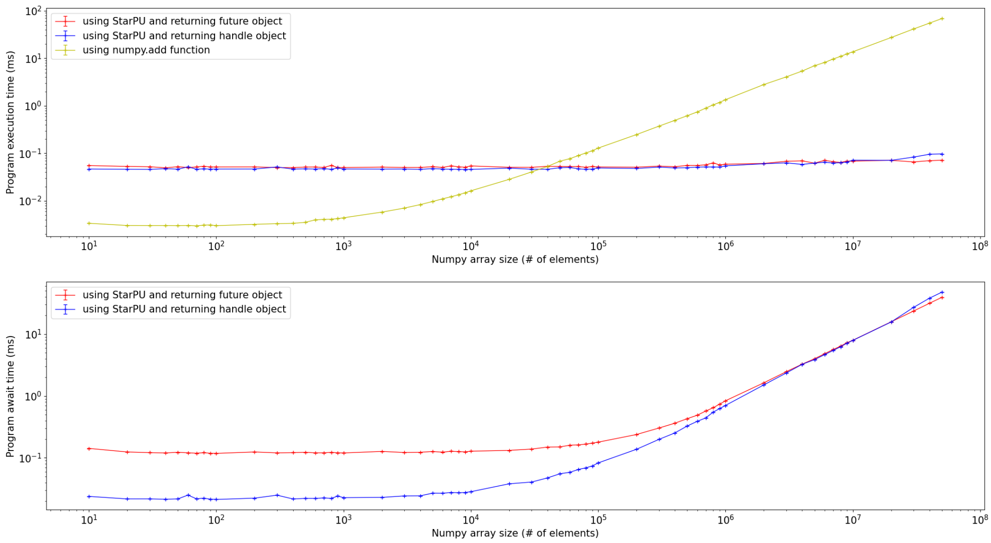
You will get the error message:

```
starpupy.error: Should not register the same mutable python object once more.
```

As performances, we showed in Section Submit Python Objects Supporting The Buffer Protocol, we add one case to compare with the others two cases. We still test the `numpy` addition (`numpy.add` running the script `test_handle_perf.sh`) with different array sizes (10, 20, ..., 100, 200, ..., 1000, 2000, ..., 10000, 20000, ..., 100000, 200000, ..., 1000000, 2000000, ..., 10000000, ..., 50000000). Three cases are:

1. Using StarPU and returning future object,
2. Using StarPU and returning handle object,
3. Without using StarPU tasks, but directly calling the `numpy.add` function.

The first plot compares the task submission time when using StarPU either returning a Future or a handle object and the program execution time without using StarPU. We can see that there is an obvious optimization using StarPU, either returning a Future or a handle object when the test array size is large. The task has not finished its execution yet as shown in second figure, the time can be used to perform other operations. When array size is not very large, returning a handle has a better execution performance than returning a Future.

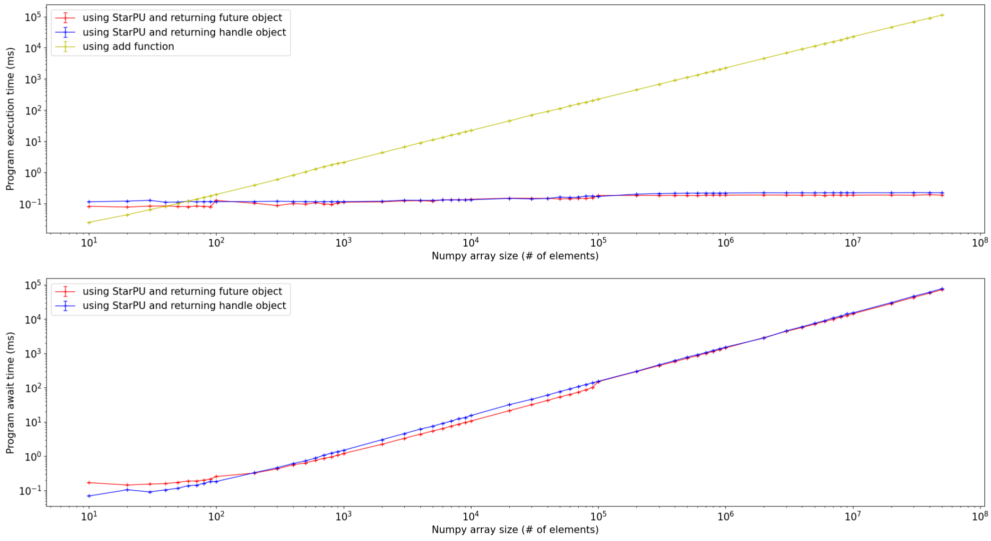


We can also define our own function to do the `numpy` operation, e.g. the element addition:

```
def add(a, b):
        for i in range(np.size(a)):
                a[i] = a[i] + b[i]
```

We will compare operation performances with the same three cases but based on our custom function `add(a, b)`.

We can see that the custom function is not as efficient as the `numpy` function overall. The optimisation for large arrays is the same when using StarPU.



30.4.2.1 Methods

As in Section Methods, the `Handle` class defines methods to provide an acquire and release mechanism.

- The method `Handle::acquire(mode)` should be called before accessing the object outside tasks (Refer to the C API starpu_data_acquire()). The access mode can be `"R"`, `"W"`, `"RW"`, the default value is "R". We will get an up-to-date copy of Python object by calling this method.

- The method `Handle::release()` must be called once the application no longer needs to access the registered data (Refer to the C API starpu_data_release()).
- The method `Handle::unregister()` to unregister the Python object handle from StarPU (Refer to the C API starpu_data_unregister()). This method will wait for all calculations to be finished before unregistering data.

The previous example can be coded as follows:

```
import numpy as np
import starpu
from starpu import Handle
starpu.init()
@starpu.access(a="RW", b="R")
def add(a,b):
   for i in range(np.size(a)):
      a[i] = a[i] + b[i]
a = np.array([1, 2, 3])
b = np.array([2, 4, 6])
a_h = Handle(a)
b_h = Handle(b)
a_h.acquire(mode = "R")
array([1, 2, 3])
a_h.release()
starpu.task_submit(ret_handle=True)(add, a_h, b_h)
a_h.acquire(mode = "R") # we get the up-to-date value

array([3, 6, 9])

a_h.release()
a_h.unregister()
```

30.4.3 Interface for Empty Numpy Array

We can register an empty `numpy` array by calling `HandleNumpy(size, type)`. The default value for `type` is `float64`.

You will find below an example which defines the function `assign` taking two arrays as parameters, the second one being an empty array which will be assigned the values of the first array.

```
import numpy as np
import starpu
from starpu import Handle
from starpu import HandleNumpy
starpu.init()
@starpu.access(b="W")
def assign(a,b):
   for i in range(min(np.size(a,0), np.size(b,0))):
      for j in range(min(np.size(a,1), np.size(b,1))):
         b[i][j] = a[i][j]
   return b
a = np.array([[1, 2, 3], [4, 5, 6]])
a_h = Handle(a)
e_h = HandleNumpy((5,10), a.dtype)
res = starpu.task_submit(ret_handle=True)(assign, a_h, e_h)
e_h.acquire()

array([[1, 2, 3, 0, 0, 0, 0, 0, 0, 0],
       [4, 5, 6, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])

e_h.release()
```

30.4.4 Array Partitioning

A n-dim `numpy` array can be split into several sub-arrays by calling the method `Handle::partition(nchildren, dim, chunks_list)` (Refer to the C API starpu_data_partition_plan()).

- `nchildren` is the number of sub-handles,
- `dim` is the dimension that we want to partition along, it can be 0 for vertical dimension, 1 for horizontal dimension, 2 for depth dimension, 3 for time dimension, ...etc.
- `chunks_list` is a list containing the size of each segment. The total length of segments in this list must be equal to the length of the selected dimension.

The method will return a sub-handle list, each of the sub-handles can be used when submitting a task with `task_submit()`. This allows to process an array in parallel, once the execution of each sub-handle is finished, the result will be directly reflected in the original n-dim array.
When the sub-handles are no longer needed, the method `Handle::unpartition(handle_list, nchildren)` should be called to clear the partition and unregister all the sub-handles (Refer to the C API starpu_data_partition_clean()).

- `handle_list` is the sub-handle list which was previously returned by the method `Handle::partition()`,
- `nchildren` is the number of sub-handles.

Here is an example to use these methods.

```
import numpy as np
import starpu
from starpu import Handle
starpu.init()
@starpu.access(a="RW", b="R")
def add(a,b):
   np.add(a,b,out=a)
n, m = 20, 10
arr = np.arange(n*m).reshape(n, m)
arr_h = Handle(arr)
arr_h.acquire(mode='RW')

 [[  0   1   2   3   4   5   6   7   8   9]
 [ 10  11  12  13  14  15  16  17  18  19]
 [ 20  21  22  23  24  25  26  27  28  29]
 [ 30  31  32  33  34  35  36  37  38  39]
 [ 40  41  42  43  44  45  46  47  48  49]
 [ 50  51  52  53  54  55  56  57  58  59]
 [ 60  61  62  63  64  65  66  67  68  69]
 [ 70  71  72  73  74  75  76  77  78  79]
 [ 80  81  82  83  84  85  86  87  88  89]
 [ 90  91  92  93  94  95  96  97  98  99]
 [100 101 102 103 104 105 106 107 108 109]
 [110 111 112 113 114 115 116 117 118 119]
 [120 121 122 123 124 125 126 127 128 129]
 [130 131 132 133 134 135 136 137 138 139]
 [140 141 142 143 144 145 146 147 148 149]
 [150 151 152 153 154 155 156 157 158 159]
 [160 161 162 163 164 165 166 167 168 169]
 [170 171 172 173 174 175 176 177 178 179]
 [180 181 182 183 184 185 186 187 188 189]
 [190 191 192 193 194 195 196 197 198 199]]

arr_h.release()
split_num = 3
arr_h_list = arr_h.partition(split_num, 1, [3,2,5]) # split into 3 sub-handles, and partition along the
      horizontal dimension
for i in range(split_num):
   res=starpu.task_submit(ret_handle=True)(add, arr_h_list[i], arr_h_list[i])
arr_h.acquire(mode='RW')

[[  0   2   4  12  16  40  48  56  64  72]
 [ 80  88  96 104 112 120 128 136 144 152]
 [160 168 176 184 192 200 208 216 224 232]
 [240 248 256 264 272 280 288 296 304 312]
 [320 328 336 172 176 180 184 188 192 196]
 [200 204 208 212 216 220 224 228 232 236]
 [120 122 124 126 128 130 132 134 136 138]
 [140 142 144 146 148 150 152 154 156 158]
 [160 162 164 166 168 170 172 174 176 178]
 [180 182 184 186 188 190 192 194 196 198]
 [200 202 204 206 208 105 106 107 108 109]
 [110 111 112 113 114 115 116 117 118 119]
 [120 121 122 123 124 125 126 127 128 129]
 [130 131 132 133 134 135 136 137 138 139]
 [140 141 142 143 144 145 146 147 148 149]
 [150 151 152 153 154 155 156 157 158 159]
 [160 161 162 163 164 165 166 167 168 169]
 [170 171 172 173 174 175 176 177 178 179]
 [180 181 182 183 184 185 186 187 188 189]
 [190 191 192 193 194 195 196 197 198 199]]
```

```
arr_h.release()
arr_h.unpartition(arr_h_list, split_num)
arr_h.unregister()
```

The method `Handle::get_partition_size(handle_list)` can be used to get the array size of each sub-array.

```
arr_h_list = arr_h.partition(split_num, 1, [3,2,5])
arr_h.get_partition_size(arr_h_list)

[60, 40, 100]
```

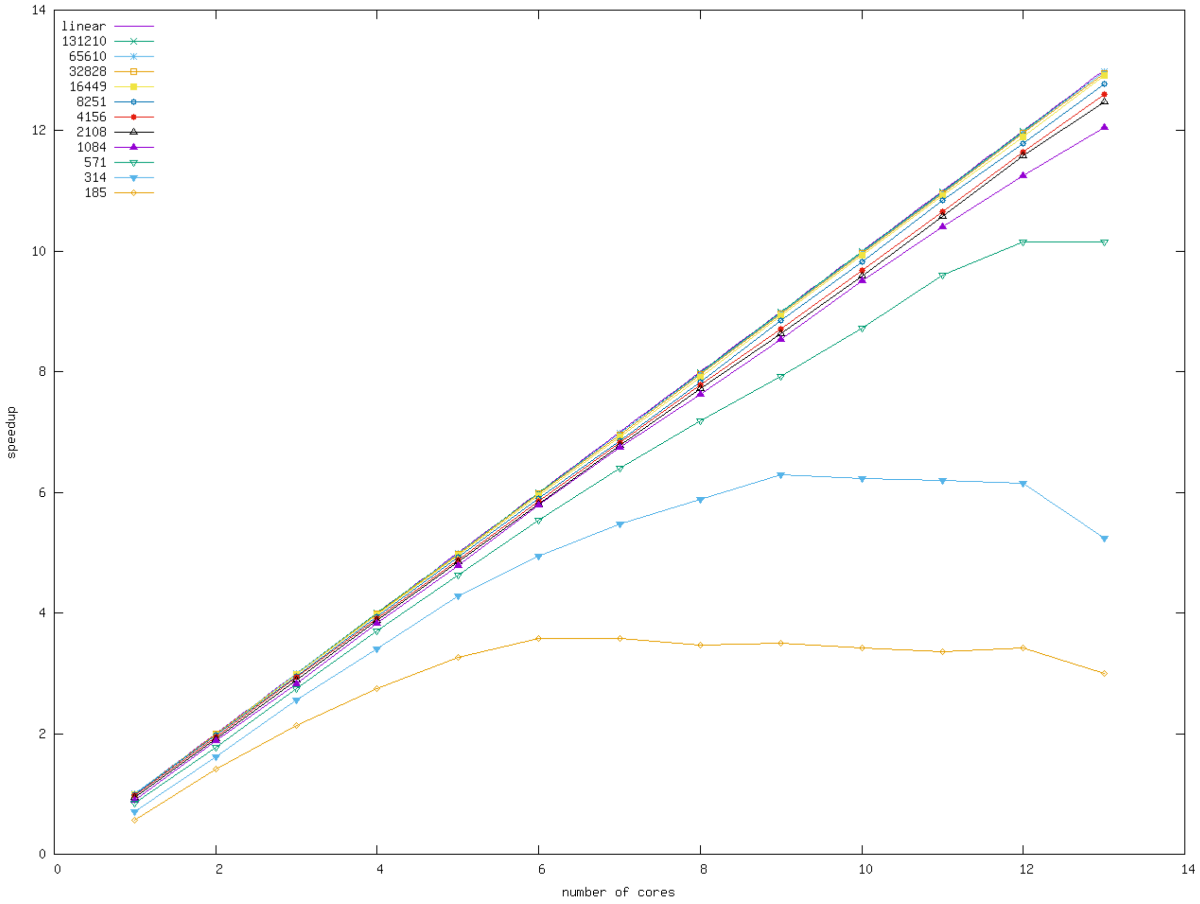
The full script is available in `starpupy/examples/starpu_py_partition.py`.

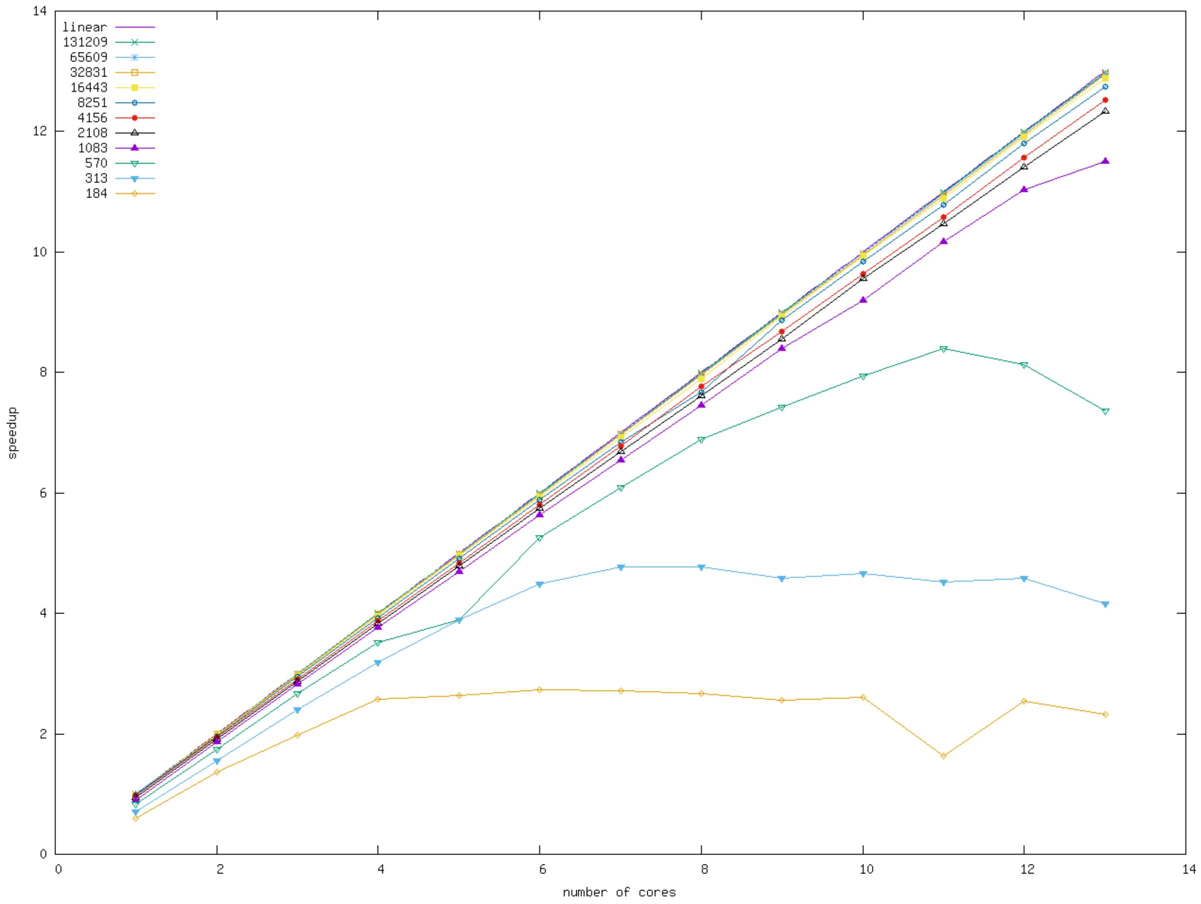
30.5 Benchmark

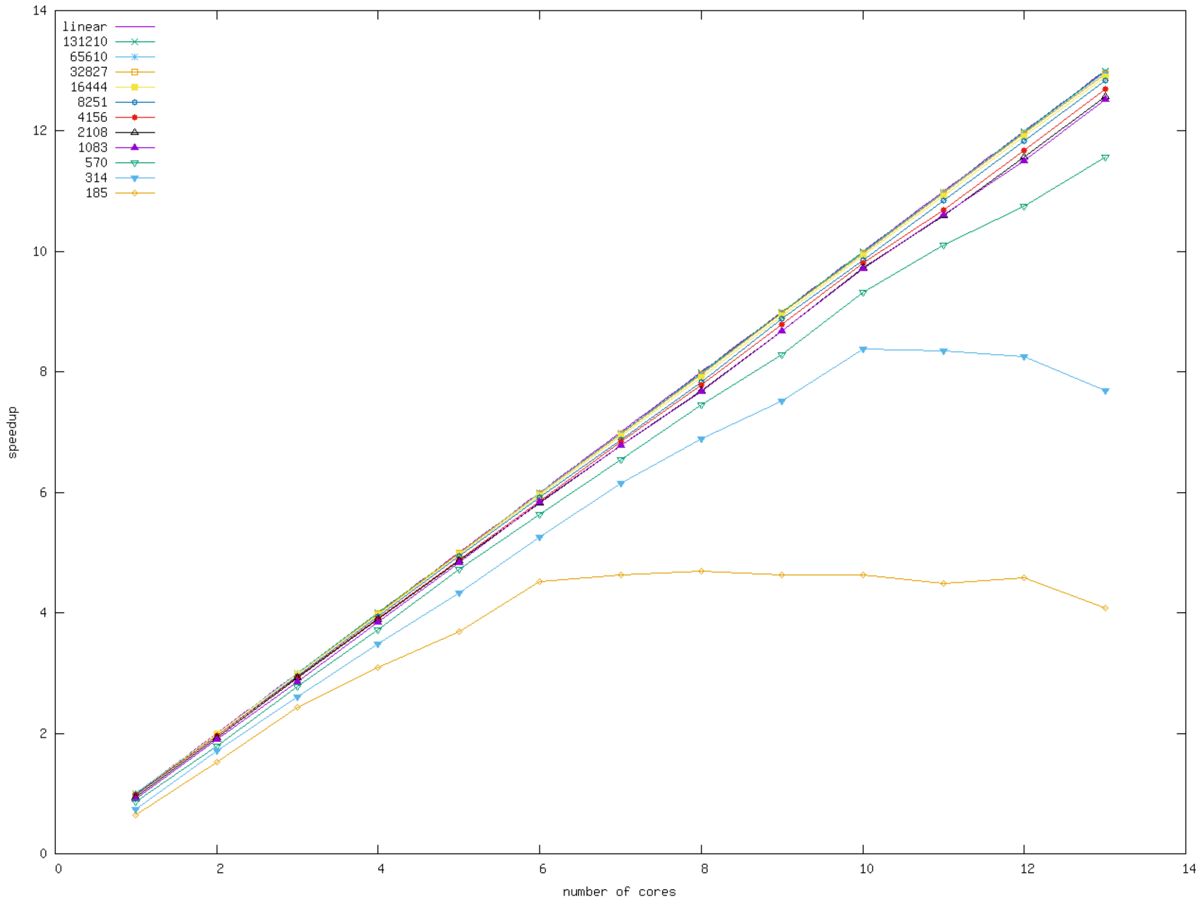
This benchmark gives a glimpse into how long a task should be (in µs) for the StarPU Python interface overhead to be low enough to keep efficiency. Running `starpupy/benchmark/tasks_size_overhead.sh` generates a plot of the speedup of tasks of various sizes, depending on the number of CPUs being used.

In the first figure, the return value is a handle object. In the second figure, the return value is a future object. In the third figure, the return value is `None`.

For example, in the figure of returning handle object, for a 571 µs task (the green line), StarPU overhead is low enough to guarantee a good speedup if the number of CPUs is not more than 12. But with the same number of CPUs, a 314 µs task (the blue line) cannot have a correct speedup. We need to decrease the number of CPUs to about 8 if we want to keep efficiency.




linear
131209
65609
32831
16443
8251
4156
2108
1083
570
313
184
speedup
number of cores
0
2
4
6
8
10
12
14


linear
131210
65610
32827
16444
8251
4156
2108
1083
570
314
185
speedup
number of cores
0
2
4
6
8
10
12
14

30.6 Running Python Functions as Pipeline Jobs (Imitating Joblib Library)

The StarPU Python interface also provides parallel computing for loops using multiprocessing, similarly to the `Joblib Library` that can simply turn out Python code into parallel computing code and thus increase the computing speed.

30.6.1 Examples

- The most basic usage is to parallelize a simple iteration.

```
from math import log10
[log10(10 ** i) for i in range(10)]
```

```
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
```

  In order to spread it over several CPUs, you need to import the `starpu.joblib` module, and use its `Parallel` class:

```
import starpu.joblib
from math import log10
starpu.init()
starpu.joblib.Parallel(n_jobs=2)(starpu.joblib.delayed(log10)(10**i)for i in range(10))
```

```
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
```

  It is also possible to first create an object of the `Parallel` class, and then call `starpu.joblib.delayed` to execute the generator expression.

```
import starpu.joblib
from math import log10
starpu.init()
parallel=starpu.joblib.Parallel(n_jobs=2)
parallel(starpu.joblib.delayed(log10)(10**i)for i in range(10))
```

```
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
```

- Instead of a generator expression, a list of functions can also be submitted as a task through the `Parallel` class.

```
import starpu.joblib
starpu.init()
#generate a list to store functions
g_func=[]
#function no input no output print hello world
def hello():
    print ("Example 1:  Hello, world!")
g_func.append(starpu.joblib.delayed(hello)())
#function has 2 int inputs and 1 int output
def multi(a, b):
    res_multi = a*b
    print("Example 2:  The result of ",a,"*",b,"is",res_multi)
    return res_multi
g_func.append(starpu.joblib.delayed(multi)(2, 3))
#function has 4 float inputs and 1 float output
def add(a, b, c, d):
    res_add = a+b+c+d
    print("Example 3:  The result of ",a,"+",b,"+",c,"+",d,"is",res_add)
    return res_add
g_func.append(starpu.joblib.delayed(add)(1.2, 2.5, 3.6, 4.9))
#function has 2 int inputs 1 float input and 1 float output 1 int output
def sub(a, b, c):
    res_sub1 = a-b-c
    res_sub2 = a-b
    print ("Example 4:  The result of ",a,"-",b,"-",c,"is",res_sub1,"and the result
of",a,"-",b,"is",res_sub2)
    return res_sub1, res_sub2
g_func.append(starpu.joblib.delayed(sub)(6, 2, 5.9))
#input is iterable function list
starpu.joblib.Parallel(n_jobs=2)(g_func)
```

  Execution:

```
Example 3: The result of  1.2 + 2.5 + 3.6 + 4.9 is 12.200000000000001
Example 1: Hello, world!
Example 4: The result of  6 - 2 - 5.9 is -1.9000000000000004 and the result of 6 - 2 is 4
Example 2: The result of  2 * 3 is 6
[None, 6, 12.200000000000001, (-1.9000000000000004, 4)]
```

- The function can also take array parameters.

```
import starpu.joblib
import numpy as np
starpu.init()
def multi_array(a, b):
        for i in range(len(a)):
                a[i] = a[i]*b[i]
A = np.arange(10)
B = np.arange(10, 20, 1)
starpu.joblib.Parallel(n_jobs=2)(starpu.joblib.delayed(multi_array)((i for i in A), (j for j in B)))
A
```

  Here the array `A` has not been modified.

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

  If we pass `A` directly as an argument, its value is updated

```
starpu.joblib.Parallel(n_jobs=2)(starpu.joblib.delayed(multi_array)(A, B))
A
```

```
array([ 0, 11, 24, 39, 56, 75, 96, 119, 144, 171])
```

  In the next call, the value of `A` is also updated.

```
starpu.joblib.Parallel(n_jobs=2)(starpu.joblib.delayed(multi_array)(b=(j for j in B), a=A))
A
```

```
array([ 0, 121, 288, 507, 784, 1125, 1536, 2023, 2592, 3249])
```

  The above three writing methods are equivalent and their execution time are very close. However, when using directly a `numpy` arrays, its value will be updated, this does not happen when generators are provided. When using a `numpy` array, it will be handled by StarPU with a data interface.

- Here an example mixing scalar objects and `numpy` arrays or generator expressions.

```
import starpu.joblib
import numpy as np
starpu.init()
def scal(a, t):
    for i in range(len(t)):
        t[i] = t[i]*a
A = np.arange(10)
starpu.joblib.Parallel(n_jobs=2)(starpu.joblib.delayed(scal)(2, (i for i in A)))
starpu.joblib.Parallel(n_jobs=2)(starpu.joblib.delayed(scal)(2,A))
```

  Again, the value of `A` is modified by the 2nd call.

```
A
```

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

The full script is available in `starpupy/examples/starpu_py_parallel.py`.

30.6.2 Parallel Parameters

The `starpu.joblib.Parallel` class accepts the following parameters:

- `mode` (string, default: `"normal"`)

  A string with the value `"normal"` or `"future"`. With the `"normal"` mode, you can call `starpu.joblib.Parallel` directly without using the `asyncio` module, and you will get the result when the task is executed. With the `"future"` mode, when calling `starpu.joblib.Parallel`, you will get a Future object as a return value. By setting the parameter `end_msg`, the given message will be displayed when the result is ready, then you can call `await` to get the result. The `asyncio` module should be imported in this case.

```
import starpu
import asyncio
from math import log10
starpu.init()
fut = starpu.joblib.Parallel(mode="future", n_jobs=3, end_msg="The result is
ready!")(starpu.joblib.delayed(log10)(10**i)for i in range(10))
The result is ready!  <_GatheringFuture finished result=[[0.0, 1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0,
8.0, 9.0]]>
await fut
```

```
[[0.0, 1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]]
```

- `end_msg` (string, default: `None`)

  A message that will be displayed when the task is executed and the result is ready. When the parameter is unset, no message will be displayed when the result is ready. In any case, you need to perform awaiting to get the result.

- `n_jobs` (int, default: `None`)

  The maximum number of concurrently running jobs. If -1 all CPUs are used. If 1 is given, no parallel computing code is used at all, which is useful for debugging. For `n_jobs` below -1, (`n_cpus` + 1 + `n_jobs`) are used. Thus, for `n_jobs` = -2, all CPUs but one are used. `None` is a marker for 'unset' that will be interpreted as `n_jobs=1` (sequential execution). `n_cpus` is the number of CPUs detected by StarPU on the running device.

- `perfmodel` (string, default : `None`)

  Set the name of the performance model. This name will be used as the filename where the performance model information will be saved. After the task is executed, one can call the function `starpu.perfmodel_plot()` by giving the symbol of perfmodel to view its performance curve.

30.6.3 Performances

- We compare the performances of the two methods for passing arguments to `the` starpu.joblib.delayed function. The first method defines a function that contains only scalars calculations, and then we pass a generator expression as an argument. The second method defines a function that contains arrays calculations, and then we pass either `numpy` arrays or generators as arguments. The second method takes less time.

```
import starpu.joblib
import numpy as np
import time
starpu.init()
N=1000000
def multi(a,b):
    res_multi = a*b
    return res_multi
print("--First method")
A = np.arange(N)
B = np.arange(N, 2*N, 1)
start_exec1 = time.time()
start_cpu1 = time.process_time()
starpu.joblib.Parallel(n_jobs=-1)(starpu.joblib.delayed(multi)(i,j) for i,j in zip(A,B))
end_exec1 = time.time()
end_cpu1 = time.process_time()
print("the program execution time is", end_exec1-start_exec1)
print("the cpu execution time is", end_cpu1-start_cpu1)
def multi_array(a, b):
    for i in range(len(a)):
        a[i] = a[i]*b[i]
    return a
print("--Second method with Numpy arrays")
A = np.arange(N)
B = np.arange(N, 2*N, 1)
start_exec2 = time.time()
start_cpu2 = time.process_time()
starpu.joblib.Parallel(n_jobs=-1)(starpu.joblib.delayed(multi_array)(A, B))
end_exec2 = time.time()
end_cpu2 = time.process_time()
print("the program execution time is", end_exec2-start_exec2)
print("the cpu execution time is", end_cpu2-start_cpu2)
print("--Second method with generators")
A = np.arange(N)
B = np.arange(N, 2*N, 1)
start_exec3 = time.time()
start_cpu3 = time.process_time()
starpu.joblib.Parallel(n_jobs=-1)(starpu.joblib.delayed(multi_array)((i for i in A), (j for j in B)))
end_exec3 = time.time()
end_cpu3 = time.process_time()
print("the program execution time is", end_exec3-start_exec3)
print("the cpu execution time is", end_cpu3-start_cpu3)
```

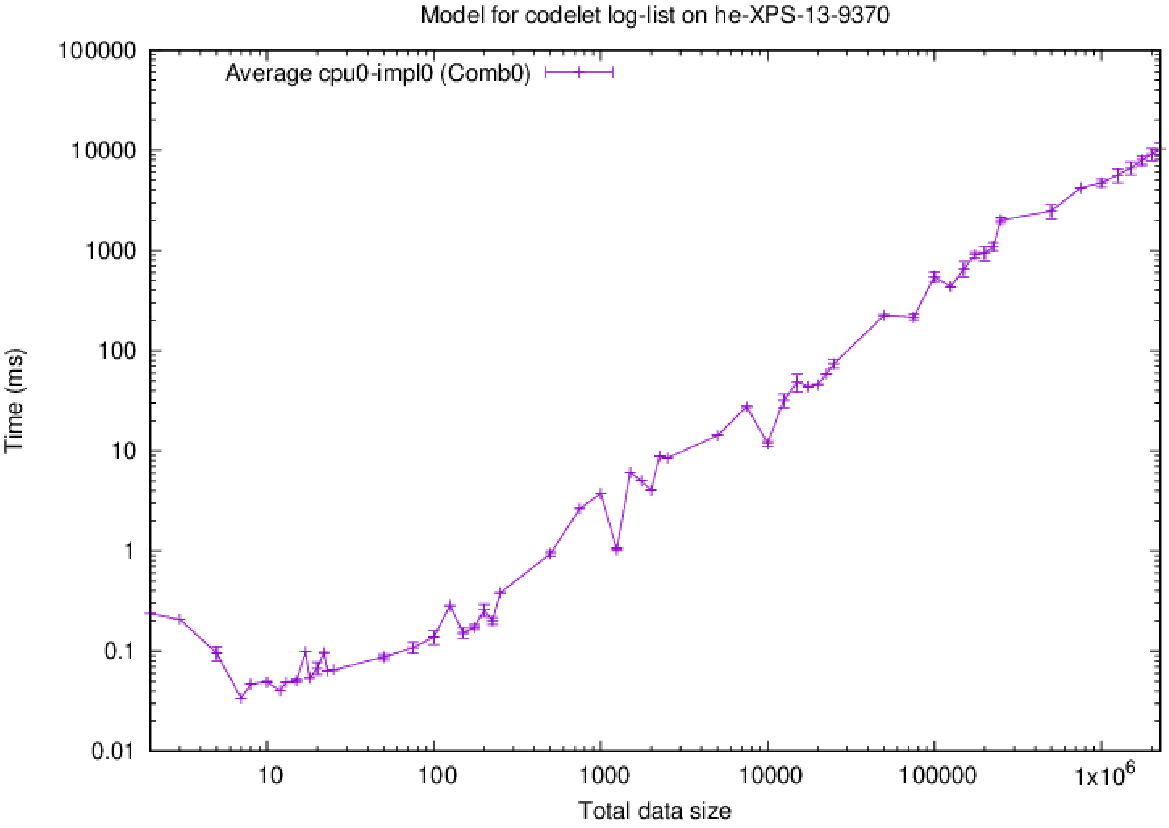
  Execution:

```
--First method
the program execution time is 3.000865936279297
the cpu execution time is 5.17138062
--Second method with Numpy arrays
the program execution time is 0.7571873664855957
the cpu execution time is 0.9166007309999991
--Second method with generators
```

```
the program execution time is 0.7259719371795654
the cpu execution time is 1.1182918959999988
```

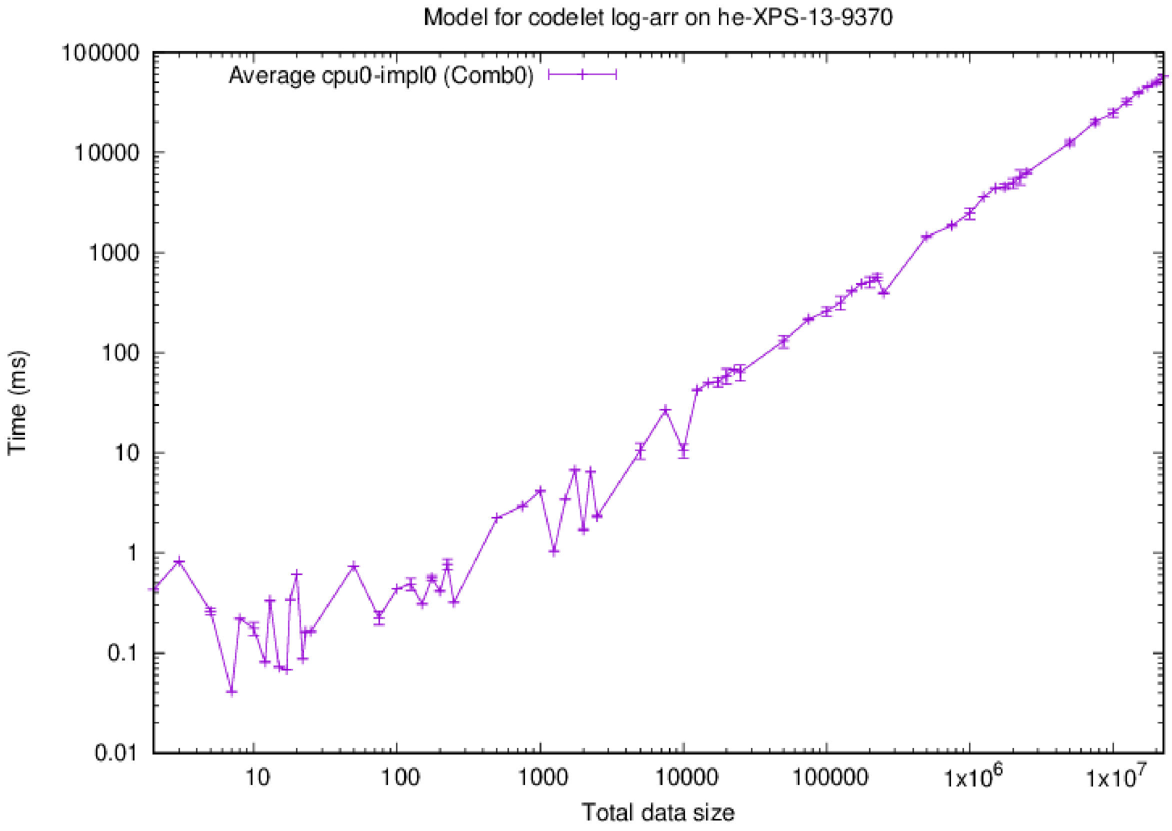
- Performance can also be shown with the performance model. Here an example with the function `log10`.

```
from math import log10
for x in [10, 100, 1000, 10000, 100000, 1000000]:
    for X in range(x, x*10, x):
        starpu.joblib.Parallel(n_jobs=-1, perfmodel="log_list")(starpu.joblib.delayed(log10)(i+1)for i
in range(X))
starpu.perfmodel_plot(perfmodel="log_list")
```



If we use a `numpy` array as parameter, the calculation can withstand larger size, as shown below.

```
from math import log10
def log10_arr(t):
    for i in range(len(t)):
        t[i] = log10(t[i])
    return t
for x in [10, 100, 1000, 10000, 100000, 1000000, 10000000]:
    for X in range(x, x*10, x):
        A = np.arange(1,X+1,1)
        starpu.joblib.Parallel(n_jobs=-1, perfmodel="log_arr")(starpu.joblib.delayed(log10_arr)(A))
starpu.perfmodel_plot(perfmodel="log_arr")
```



30.7 Multiple Interpreters

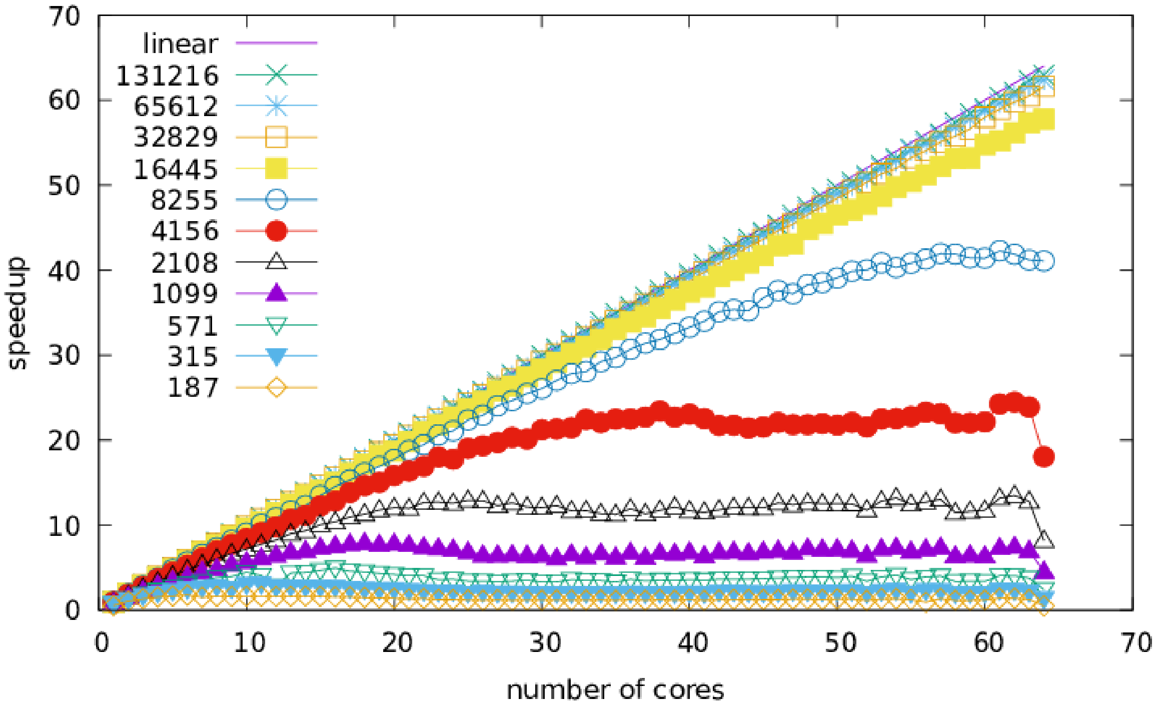
It is possible to use multiple interpreters when running python applications. To do so, you need to set the variable STARPUPY_MULTI_INTERPRETER when running a StarPU Python application.

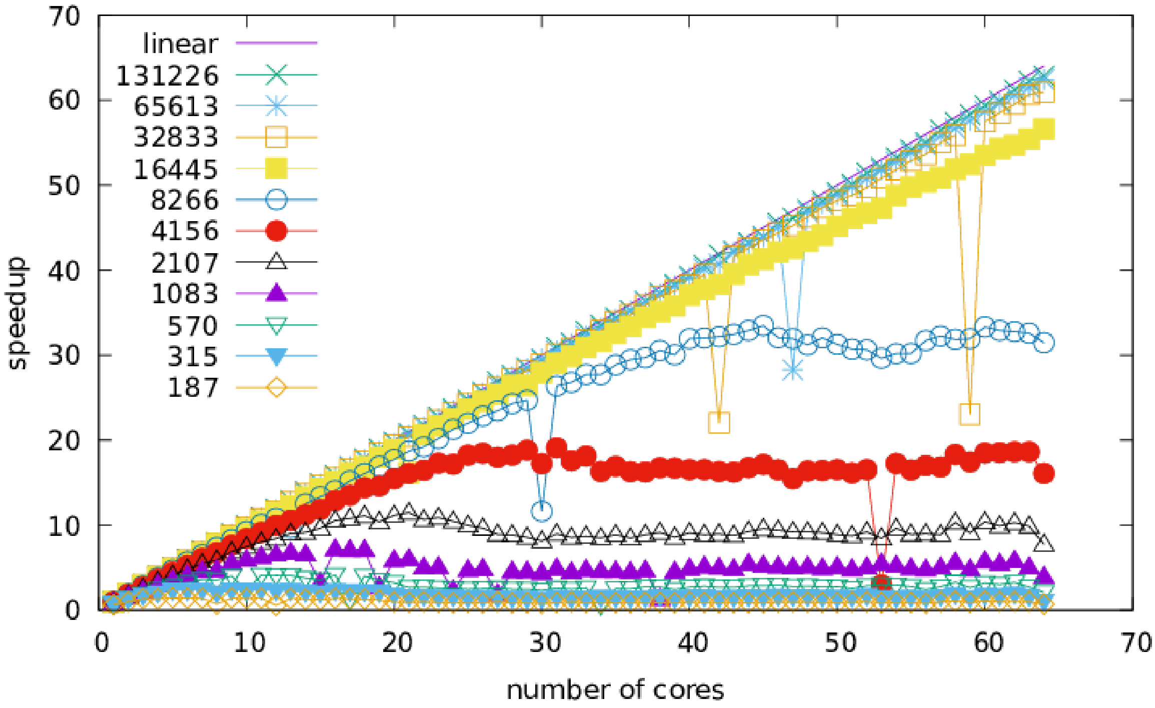
Python interpreters share the Global Interpreter Lock (GIL), which requires that at any time, one and only one thread has the right to execute a task. In other words, GIL makes the multiple interpreters execution of Python actually serial rather than parallel, and the execution of Python program is single-threaded essentially. Therefore, if the application is pure Python script, even with multi-interpreters, the program cannot be executed in parallel, unless an external C application is called.

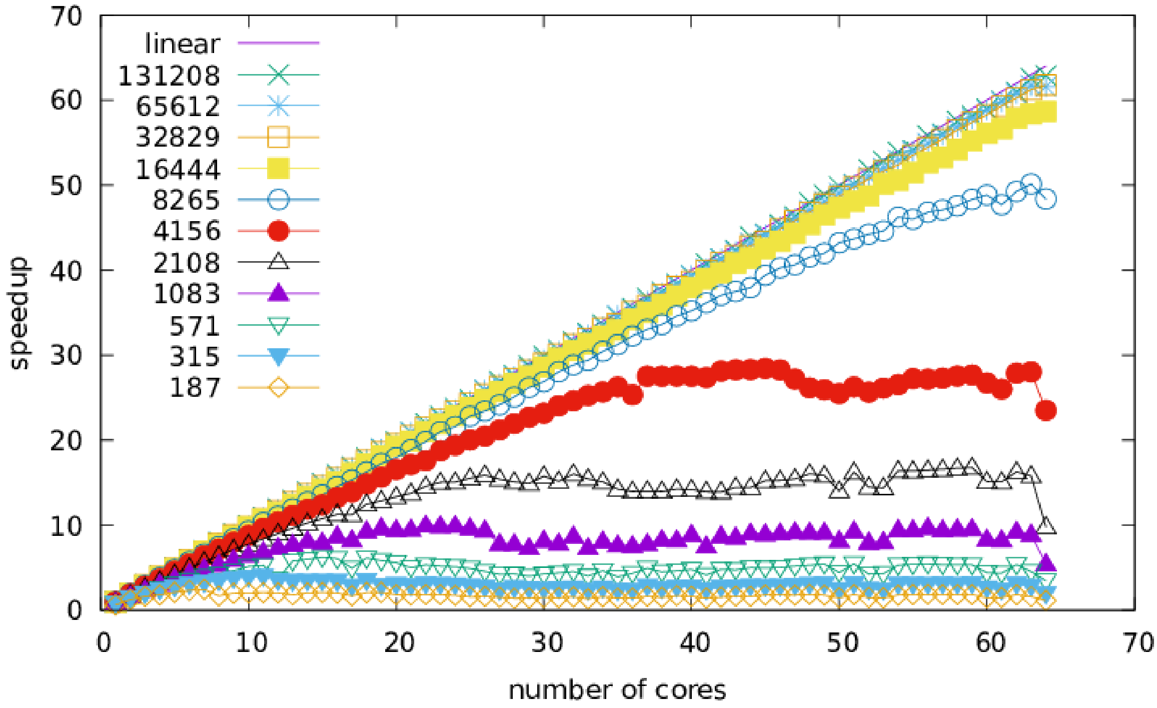
Fortunately now there is a quite positive development. Python developers are preparing to implement stop sharing the GIL between interpreters (`https://peps.nogil.dev/pep-0684/`) or even make GIL optional so that Python code can be run without GIL (`https://peps.nogil.dev/pep-0701/`), that will facilitate true parallelism with the next Python version.

In order to transfer data between interpreters, the module `cloudpickle` is used to serialize Python objects in contiguous byte array. This mechanism increases the overhead of the StarPU Python interface, as shown in the following plots, to be compared to the plots given in Benchmark.

In the first figure, the return value is a handle object. In the second figure, the return value is a future object. In the third figure, the return value is `None`.


speedup
number of cores
0
10
20
30
40
50
60
70
linear
131216
65612
32829
16445
8255
4156
2108
1099
571
315
187


speedup
number of cores
0
10
20
30
40
50
60
70
linear
131226
65613
32833
16445
8266
4156
2107
1083
570
315
187

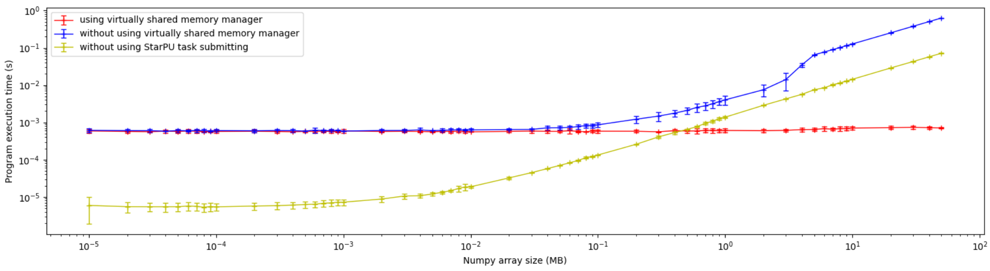


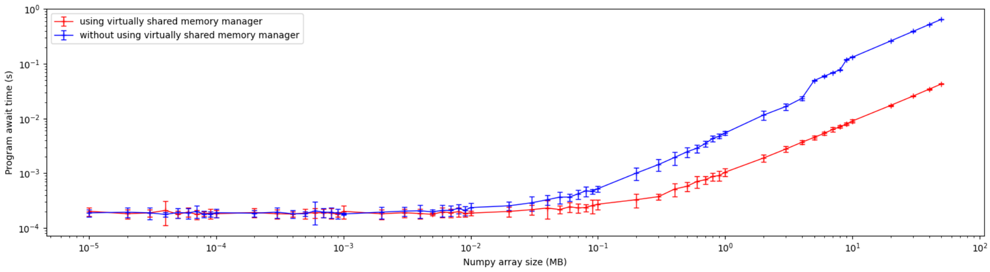
In order to reflect this influence more intuitively, we make a performance comparison.

By default, StarPU uses virtually shared memory manager for Python objects supporting buffer protocol that allows to minimize data transfers. But in the case of multi-interpreter, if we do not use virtually shared memory manager, data transfer can be realized only with the help of cloudpickle.

We will show the operation performances below (Running `test_handle_perf_pickle.sh`). The operation that we test is `numpy` addition (`numpy.add`), and the array size is 10, 20, ..., 100, 200, ..., 1000, 2000, ..., 10000, 2000, ..., 100000,200000, ..., 1000000, 2000000, ..., 10000000, ..., 50000000. We compared three cases: first, using virtually shared memory manager, second, without using virtually shared memory manager, third, without using StarPU task submitting, but directly calling `numpy.add` function.

In the first figure, we compare the submission time when using StarPU and the execution time without using StarPU. We can see that there is still an obvious optimization using StarPU virtually shared memory manager when the test array size is large. However, if only using cloudpickle, StarPU Python interface cannot provide an effective optimization. And in the second figure, we can see that the same operation will take more time to finish the program execution when only using cloudpickle.

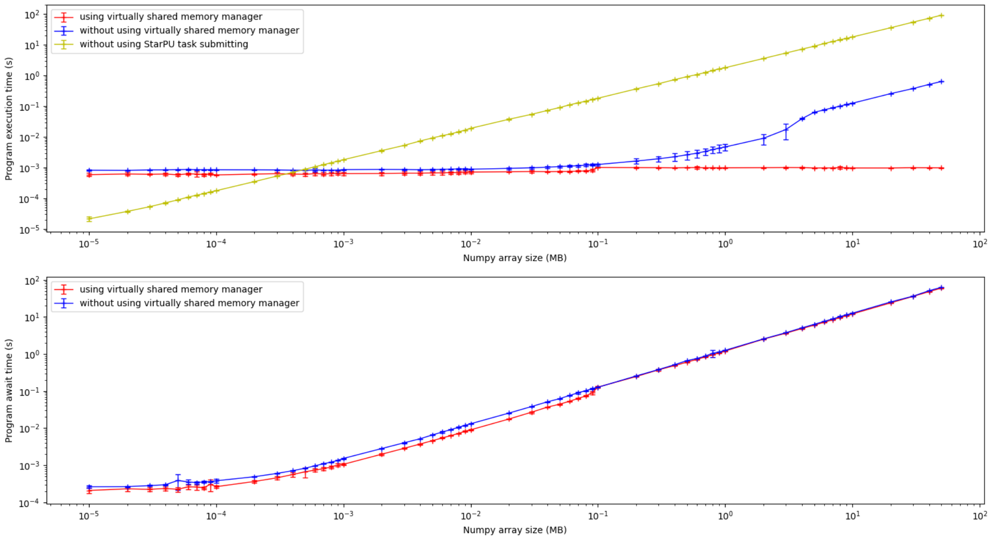




We can also define our own function to do the `numpy` operation, e.g. the element addition:

```
def add(a, b):
    for i in range(np.size(a)):
        a[i] = a[i] + b[i]
```

We will compare operation performances of the same three cases, but based on the custom function `add(a, b)`. We can see that the custom function takes more time than `numpy` function overall. Although the same operation still takes more time to submit the task when only using cloudpickle than with virtually shared memory manager, there is still a better optimization. The operation takes less time than only calling a custom function even when the array is not very large.



30.8 Master Slave Support

StarPU Python interface provides MPI master slave support as well. Please refer to MPI Master Slave Support for the specific usage.

When you write your Python script, make sure to import all required functions before the `starpu` module. Functions imported after the `starpu` module can only be submitted using their name as a string when calling `task_submit()`, this will decrease the submission efficiency.

(TODO)

30.9 StarPUPY and Simgrid

In simgrid mode, the Python interpreter will not be aware of simgrid and will thus not notify it when some thread is blocked waiting for something to happen in another thread. This notably means that the `asyncio` mode and waiting for a `future` will not work, and one thus has to use StarPUPY-provided functions to wait for completion, such as `starpupy.task_wait_for_all()` or `data.acquire`.

Also, we have not yet implemented not calling the actual call of the task function, so the execution time will be longer than in real execution, since not only it executes computations, but also sequentially, and adds the simulation overhead.

Chapter 31

The StarPU OpenMP Runtime Support (SORS)

StarPU provides the necessary routines and support to implement an OpenMP (`http://www.openmp.org/`) runtime compliant with the revision 3.1 of the language specification, and compliant with the task-related data dependency functionalities introduced in the revision 4.0 of the language. This StarPU OpenMP Runtime Support (SORS) has been designed to be targeted by OpenMP compilers such as the Klang-OMP compiler. Most supported OpenMP directives can both be implemented inline or as outlined functions.
All functions are defined in OpenMP Runtime Support.
Several examples supporting OpenMP API are provided in StarPU's `tests/openmp/` directory.

31.1 Implementation Details and Specificities

31.1.1 Main Thread

When using SORS, the main thread gets involved in executing OpenMP tasks just like every other threads, in order to be compliant with the specification execution model. This contrasts with StarPU's usual execution model, where the main thread submit tasks but does not take part in executing them.

31.1.2 Extended Task Semantics

The semantics of tasks generated by SORS are extended with respect to regular StarPU tasks in that SORS' tasks may block and be preempted by SORS call, whereas regular StarPU tasks cannot. SORS tasks may coexist with regular StarPU tasks. However, only the tasks created using SORS API functions inherit from extended semantics.

31.2 Configuration

SORS can be compiled into `libstarpu` through the `configure` option --enable-openmp. Conditional compiled source codes may check for the availability of the OpenMP Runtime Support by testing whether the C preprocessor macro `STARPU_OPENMP` is defined or not.

31.3 Initialization and Shutdown

SORS needs to be executed/terminated by the starpu_omp_init() / starpu_omp_shutdown() instead of starpu_init() / starpu_shutdown(). This requirement is necessary to make sure that the main thread gets the proper execution environment to run OpenMP tasks. These calls will usually be performed by a compiler runtime. Thus, they can be executed from a constructor/destructor such as this:

```
__attribute__((constructor))
static void omp_constructor(void)
{
        int ret = starpu_omp_init();
        STARPU_CHECK_RETURN_VALUE(ret, "starpu_omp_init");
}
__attribute__((destructor))
static void omp_destructor(void)
{
        starpu_omp_shutdown();
}
```

Basic examples are available in the files `tests/openmp/init_exit_01.c` and `tests/openmp/init_exit_02.c`.

See also

starpu_omp_init()

starpu_omp_shutdown()

31.4 Parallel Regions and Worksharing

SORS provides functions to create OpenMP parallel regions, as well as mapping work on participating workers. The current implementation does not provide nested active parallel regions: Parallel regions may be created recursively, however only the first level parallel region may have more than one worker. From an internal point-of-view, SORS' parallel regions are implemented as a set of implicit, extended semantics StarPU tasks, following the execution model of the OpenMP specification. Thus, SORS' parallel region tasks may block and be preempted, by SORS calls, enabling constructs such as barriers.

31.4.1 Parallel Regions

Parallel regions can be created with the function starpu_omp_parallel_region() which accepts a set of attributes as parameter. The execution of the calling task is suspended until the parallel region completes. The field starpu_omp_parallel_region_attr::cl is a regular StarPU codelet. However, only CPU codelets are supported for parallel regions. Here is an example of use:

```
void parallel_region_f(void *buffers[], void *args)
{
        (void) buffers;
        (void) args;
        pthread_t tid = pthread_self();
        int worker_id = starpu_worker_get_id();
        printf("[tid %p] task thread = %d\n", (void *)tid, worker_id);
}
void f(void)
{
        struct starpu_omp_parallel_region_attr attr;
        memset(&attr, 0, sizeof(attr));
        attr.cl.cpu_funcs[0] = parallel_region_f;
        attr.cl.where        = STARPU_CPU;
        attr.if_clause       = 1;
        starpu_omp_parallel_region(&attr);
        return 0;
}
```

A basic example is available in the file `tests/openmp/parallel_01.c`.

See also

struct starpu_omp_parallel_region_attr

starpu_omp_parallel_region()

31.4.2 Parallel For

OpenMP `for` loops are provided by the starpu_omp_for() group of functions. Variants are available for inline or outlined implementations. SORS supports `static`, `dynamic`, and `guided` loop scheduling clauses. The `auto` scheduling clause is implemented as `static`. The `runtime` scheduling clause honors the scheduling mode selected through the environment variable `OMP_SCHEDULE` or the starpu_omp_set_schedule() function. For loops with the `ordered` clause are also supported. An implicit barrier can be enforced or skipped at the end of the worksharing construct, according to the value of the `nowait` parameter.

The canonical family of starpu_omp_for() functions provide each instance with the first iteration number and the number of iterations (possibly zero) to perform. The alternate family of starpu_omp_for_alt() functions provide each instance with the (possibly empty) range of iterations to perform, including the first and excluding the last. An example is available in the file `tests/openmp/parallel_for_01.c`.

The family of starpu_omp_ordered() functions enable to implement OpenMP's ordered construct, a region with a parallel for loop that is guaranteed to be executed in the sequential order of the loop iterations. An example is available in the file `tests/openmp/parallel_for_ordered_01.c`.

```
void for_g(unsigned long long i, unsigned long long nb_i, void *arg)
{
        (void) arg;
        for (; nb_i > 0; i++, nb_i--)
        {
                array[i] = 1;
        }
}
void parallel_region_f(void *buffers[], void *args)
{
        (void) buffers;
        (void) args;
        starpu_omp_for(for_g, NULL, NB_ITERS, CHUNK, starpu_omp_sched_static, 0, 0);
}
```

See also

starpu_omp_for()

starpu_omp_for_inline_first()

starpu_omp_for_inline_next()

starpu_omp_for_alt()

starpu_omp_for_inline_first_alt()

starpu_omp_for_inline_next_alt()

starpu_omp_ordered()

starpu_omp_ordered_inline_begin()

starpu_omp_ordered_inline_end()

31.4.3 Sections

OpenMP `sections` worksharing constructs are supported using the set of starpu_omp_sections() variants. The general principle is either to provide an array of per-section functions or a single function that will redirect the execution to the suitable per-section functions. An implicit barrier can be enforced or skipped at the end of the worksharing construct, according to the value of the `nowait` parameter.

```
void parallel_region_f(void *buffers[], void *args)
{
        (void) buffers;
        (void) args;
        section_funcs[0] = f;
        section_funcs[1] = g;
        section_funcs[2] = h;
        section_funcs[3] = i;
        section_args[0] = arg_f;
        section_args[1] = arg_g;
        section_args[2] = arg_h;
        section_args[3] = arg_i;
        starpu_omp_sections(4, section_f, section_args, 0);
}
```

An example is available in the file `tests/openmp/parallel_sections_01.c`.

See also

starpu_omp_sections()

starpu_omp_sections_combined()

31.4.4 Single

OpenMP `single` workharing constructs are supported using the set of starpu_omp_single() variants. An implicit barrier can be enforced or skipped at the end of the worksharing construct, according to the value of the `nowait` parameter. An example is available in the file `tests/openmp/parallel_single_nowait_01.c`.

```
void single_f(void *arg)
{
        (void) arg;
        pthread_t tid = pthread_self();
        int worker_id = starpu_worker_get_id();
        printf("[tid %p] task thread = %d -- single\n", (void *)tid, worker_id);
}
void parallel_region_f(void *buffers[], void *args)
{
        (void) buffers;
        (void) args;
        starpu_omp_single(single_f, NULL, 0);
}
```

SORS also provides dedicated support for `single` sections with `copyprivate` clauses through the starpu_omp_single_copyprivate() function variants. The OpenMP `master` directive is supported as well, using the starpu_omp_master() function variants. An example is available in the file `tests/openmp/parallel_single_copyprivate_01.c`.

See also

starpu_omp_master()

starpu_omp_master_inline()

starpu_omp_single()

starpu_omp_single_inline()

starpu_omp_single_copyprivate()

starpu_omp_single_copyprivate_inline_begin()

starpu_omp_single_copyprivate_inline_end()

31.5 Tasks

SORS implements the necessary support of OpenMP 3.1 and OpenMP 4.0's so-called explicit tasks, together with OpenMP 4.0's data dependency management.

31.5.1 Explicit Tasks

Explicit OpenMP tasks are created with SORS using the starpu_omp_task_region() function. The implementation supports `if`, `final`, `untied` and `mergeable` clauses as defined in the OpenMP specification. Unless specified otherwise by the appropriate clause(s), the created task may be executed by any participating worker of the current parallel region.
The current SORS implementation requires explicit tasks to be created within the context of an active parallel region. In particular, an explicit task cannot be created by the main thread outside a parallel region. Explicit OpenMP tasks created using starpu_omp_task_region() are implemented as StarPU tasks with extended semantics, and may as such be blocked and preempted by SORS routines.
The current SORS implementation supports recursive explicit tasks creation, to ensure compliance with the OpenMP specification. However, it should be noted that StarPU is not designed nor optimized for efficiently scheduling of recursive task applications.
The code below shows how to create 4 explicit tasks within a parallel region.

```
void task_region_g(void *buffers[], void *args)
{
        (void) buffers;
        (void) args;
        pthread tid = pthread_self();
        int worker_id = starpu_worker_get_id();
        printf("[tid %p] task thread = %d:  explicit task \"g\"\n", (void *)tid, worker_id);
}
void parallel_region_f(void *buffers[], void *args)
{
        (void) buffers;
        (void) args;
        struct starpu_omp_task_region_attr attr;
        memset(&attr, 0, sizeof(attr));
        attr.cl.cpu_funcs[0]  = task_region_g;
        attr.cl.where         = STARPU_CPU;
        attr.if_clause        = 1;
        attr.final_clause     = 0;
        attr.untied_clause    = 1;
        attr.mergeable_clause = 0;
        starpu_omp_task_region(&attr);
        starpu_omp_task_region(&attr);
        starpu_omp_task_region(&attr);
        starpu_omp_task_region(&attr);
}
```

An example is available in the file `tests/openmp/parallel_01.c`.

See also

struct starpu_omp_task_region_attr

starpu_omp_task_region()

31.5.2 Data Dependencies

SORS implements inter-tasks data dependencies as specified in OpenMP 4.0. Data dependencies are expressed using regular StarPU data handles (starpu_data_handle_t) plugged into the task's `attr.cl` codelet. The family of starpu_vector_data_register() -like functions, the starpu_omp_handle_register() and starpu_omp_handle_unregister() functions, and the starpu_omp_data_lookup() function may be used to register a memory area and to retrieve the current data handle associated with a pointer respectively. The testcase `./tests/openmp/task_02.c` gives a detailed example of using OpenMP 4.0 tasks dependencies with SORS implementation.

Note: the OpenMP 4.0 specification only supports data dependencies between sibling tasks, that are tasks created by the same implicit or explicit parent task. The current SORS implementation also only supports data dependencies between sibling tasks. Consequently, the behavior is unspecified if dependencies are expressed between tasks that have not been created by the same parent task.

31.5.3 TaskWait and TaskGroup

SORS implements both the `taskwait` and `taskgroup` OpenMP task synchronization constructs specified in OpenMP 4.0, with the starpu_omp_taskwait() and starpu_omp_taskgroup() functions, respectively.

An example of starpu_omp_taskwait() use, creating two explicit tasks and waiting for their completion:

```
void task_region_g(void *buffers[], void *args)
{
        (void) buffers;
        (void) args;
        printf("Hello, World!\n");
}
void parallel_region_f(void *buffers[], void *args)
{
        (void) buffers;
        (void) args;
        struct starpu_omp_task_region_attr attr;
        memset(&attr, 0, sizeof(attr));
        attr.cl.cpu_funcs[0]  = task_region_g;
        attr.cl.where         = STARPU_CPU;
        attr.if_clause        = 1;
        attr.final_clause     = 0;
        attr.untied_clause    = 1;
        attr.mergeable_clause = 0;
        starpu_omp_task_region(&attr);
        starpu_omp_task_region(&attr);
        starpu_omp_taskwait();
```

An example is available in the file `tests/openmp/taskwait_01.c`.

An example of starpu_omp_taskgroup() use, creating a task group of two explicit tasks:

```
void task_region_g(void *buffers[], void *args)
{
        (void) buffers;
        (void) args;
        printf("Hello, World!\n");
}
void taskgroup_f(void *arg)
{
        (void)arg;
        struct starpu_omp_task_region_attr attr;
        memset(&attr, 0, sizeof(attr));
        attr.cl.cpu_funcs[0]  = task_region_g;
        attr.cl.where         = STARPU_CPU;
        attr.if_clause        = 1;
        attr.final_clause     = 0;
        attr.untied_clause    = 1;
        attr.mergeable_clause = 0;
        starpu_omp_task_region(&attr);
        starpu_omp_task_region(&attr);
}
void parallel_region_f(void *buffers[], void *args)
{
        (void) buffers;
        (void) args;
        starpu_omp_taskgroup(taskgroup_f, (void *)NULL);
}
```

An example is available in the file `tests/openmp/taskgroup_01.c`.

See also

starpu_omp_task_region()

starpu_omp_taskwait()

starpu_omp_taskgroup()

starpu_omp_taskgroup_inline_begin()

starpu_omp_taskgroup_inline_end()

31.6 Synchronization Support

SORS implements objects and method to build common OpenMP synchronization constructs.

31.6.1 Simple Locks

SORS Simple Locks are opaque starpu_omp_lock_t objects enabling multiple tasks to synchronize with each others, following the Simple Lock constructs defined by the OpenMP specification. In accordance with such specification, simple locks may not be acquired multiple times by the same task, without being released in-between; otherwise, deadlocks may result. Codes requiring the possibility to lock multiple times recursively should use Nestable Locks (NestableLock). Codes NOT requiring the possibility to lock multiple times recursively should use Simple Locks as they incur less processing overhead than Nestable Locks. An example is available in the file `tests/openmp/parallel_simple_lock_01.c`.

See also

starpu_omp_lock_t

starpu_omp_init_lock()

starpu_omp_destroy_lock()

starpu_omp_set_lock()

starpu_omp_unset_lock()

starpu_omp_test_lock()

31.6.2 Nestable Locks

SORS Nestable Locks are opaque starpu_omp_nest_lock_t objects enabling multiple tasks to synchronize with each others, following the Nestable Lock constructs defined by the OpenMP specification. In accordance with such specification, nestable locks may be acquired multiple times recursively by the same task without deadlocking. Nested locking and unlocking operations must be well parenthesized at any time, otherwise deadlock and/or undefined behavior may occur. Codes requiring the possibility to lock multiple times recursively should use Nestable Locks. Codes NOT requiring the possibility to lock multiple times recursively should use Simple Locks (SimpleLock) instead, as they incur less processing overhead than Nestable Locks. An example is available in the file `tests/openmp/parallel_nested_lock_01.c`.

See also

starpu_omp_nest_lock_t

starpu_omp_init_nest_lock()

starpu_omp_destroy_nest_lock()

starpu_omp_set_nest_lock()

starpu_omp_unset_nest_lock()

starpu_omp_test_nest_lock()

31.6.3 Critical Sections

SORS implements support for OpenMP critical sections through the family of starpu_omp_critical functions. Critical sections may optionally be named. There is a single, common anonymous critical section. Mutual exclusion only occur within the scope of single critical section, either a named one or the anonymous one. Corresponding examples are available in the files `tests/openmp/parallel_critical_01.c` and `tests/openmp/parallel_critical_inline_01.c`.

See also

starpu_omp_critical()

starpu_omp_critical_inline_begin()

starpu_omp_critical_inline_end()

31.6.4 Barriers

SORS provides the starpu_omp_barrier() function to implement barriers over parallel region teams. In accordance with the OpenMP specification, the starpu_omp_barrier() function waits for every implicit task of the parallel region to reach the barrier and every explicit task launched by the parallel region to complete, before returning. An example is available in the file `tests/openmp/parallel_barrier_01.c`.

See also

starpu_omp_barrier()

31.7 Example: An OpenMP LLVM Support

SORS has been used to implement an OpenMP LLVM Support. This allows to seamlessly run OpenMP applications on top of StarPU.
To enable this support, one just needs to call `configure` with the option --enable-openmp-llvm.
After installation, the directory `lib/starpu/examples/starpu_openmp_llvm` contains a OpenMP application, its source code and the executable compiled with the StarPU OpenMP LLVM support, as well as a README file explaining how to use the support for your own application.
One just needs to compile an OpenMP application with `clang` and to execute it the StarPU OpenMP LLVM support library file instead of the default `libomp.so`.

31.8 OpenMP Standard Functions in StarPU

StarPU provides severals functions which are very similar to their OpenMP counterparts but are adapted to the StarPU runtime system. These functions are:

- starpu_omp_set_num_threads()
- starpu_omp_get_num_threads()
- starpu_omp_get_thread_num()
- starpu_omp_get_max_threads()
- starpu_omp_get_num_procs() which is used to get the number of available StarPU CPU workers.
- starpu_omp_in_parallel()
- starpu_omp_set_dynamic()
- starpu_omp_get_dynamic()
- starpu_omp_set_nested()
- starpu_omp_get_nested()
- starpu_omp_get_cancellation()
- starpu_omp_set_schedule()
- starpu_omp_get_schedule()
- starpu_omp_get_thread_limit()
- starpu_omp_set_max_active_levels()

- starpu_omp_get_max_active_levels()
- starpu_omp_get_level()
- starpu_omp_get_ancestor_thread_num()
- starpu_omp_get_team_size()
- starpu_omp_get_active_level()
- starpu_omp_in_final()
- starpu_omp_get_proc_bind()
- starpu_omp_get_num_places()
- starpu_omp_get_place_num_procs()
- starpu_omp_get_place_proc_ids()
- starpu_omp_get_place_num()
- starpu_omp_get_partition_num_places()
- starpu_omp_get_partition_place_nums()
- starpu_omp_set_default_device()
- starpu_omp_get_default_device()
- starpu_omp_get_num_devices()
- starpu_omp_get_num_teams()
- starpu_omp_get_team_num()
- starpu_omp_is_initial_device()
- starpu_omp_get_initial_device()
- starpu_omp_get_max_task_priority()
- starpu_omp_get_wtime()
- starpu_omp_get_wtick()

Part VII

StarPU Extensions

Chapter 32

Organization

This part explains the advanced concepts of StarPU. It is intended for users whose applications need more than basic task submission.
You can learn more knowledge about some important and core concepts in StarPU:

- After reading Chapter Tasks In StarPU, you can get more information about how to manage tasks in StarPU in Chapter Advanced Tasks In StarPU.
- After reading Chapter Data Management, you can know more about how to manage the data layout of your applications in Chapter Advanced Data Management.
- After reading Chapter Scheduling, you can get some advanced scheduling policies in StarPU in Chapters Advanced Scheduling, Scheduling Contexts and Scheduling Context Hypervisor.
- Chapter How To Define A New Scheduling Policy explains how to define a StarPU task scheduling policy either in a basic monolithic way, or in a modular way.

Other chapters cover some further usages of StarPU.

- Chapters CUDA Support and OpenCL Support show how to use GPU devices with CUDA or OpenCL. Chapter Maxeler FPGA Support explains how StarPU support Field Programmable Gate Array (FPGA) applications exploiting DFE configurations.
- If you need to store more data than what the main memory (RAM) can store, Chapter Out Of Core presents how to add a new memory node on a disk and how to use it.
- Chapter MPI Support shows how to integrate MPI processes in StarPU.
- Chapter TCP/IP Support shows a TCP/IP master slave mechanism which can execute application across many remote cores without thinking about data distribution.
- Chapter Transactions shows how to cancel a sequence of already submitted tasks based on a just-in-time decision.
- Chapter Fault Tolerance explains how StarPU provide supports for failure of tasks or even failure of complete nodes.
- Chapter FFT Support explains how StarPU provides a similar library to both `fftw` and `cufft`, but by adding a support from both CPUs and GPUs.
- Chapter SOCL OpenCL Extensions explains how OpenCL applications can transparently be run using StarPU, by givings unified access to every available OpenCL device.
- We propose a hierarchical tasks model in Chapter Hierarchical DAGS to enable tasks subgraphs at runtime for a more dynamic task graph.
- You can find how to partition a machine into parallel workers in Chapter Creating Parallel Workers On A Machine.
- Chapter Interoperability Support shows how StarPU can coexist with other parallel software elements without resulting in computing core oversubscription or undersubscription.

- Chapter SimGrid Support shows you how to simulate execution on an arbitrary platform.
- Tools to help debugging applications are presented in Chapter Debugging Tools.

And finally, chapter Helpers gives a list of StarPU utility functions.

Chapter 33

Advanced Tasks In StarPU

33.1 Task Dependencies

33.1.1 Sequential Consistency

By default, task dependencies are inferred from data dependency (sequential coherency) by StarPU. The application can however disable sequential coherency for some data, and dependencies can be specifically expressed.
Setting (or unsetting) sequential consistency can be done at the data level by calling starpu_data_set_sequential_consistency_flag() for a specific data (an example is in the file `examples/dependency/task_end_dep.c`) or starpu_data_set_default_sequential for all data (an example is in the file `tests/main/subgraph_repeat.c`).
The sequential consistency mode can also be gotten by calling starpu_data_get_sequential_consistency_flag() for a specific data or get the default sequential consistency flag by calling starpu_data_get_default_sequential_consistency_flag().
Setting (or unsetting) sequential consistency can also be done at task level by setting the field starpu_task::sequential_consistency to `0` (an example is in the file `tests/main/deploop.c`).
Sequential consistency can also be set (or unset) for each handle of a specific task, this is done by using the field starpu_task::handles_sequential_consistency. When set, its value should be an array with the number of elements being the number of handles for the task, each element of the array being the sequential consistency for the `i-th` handle of the task. The field can easily be set when calling starpu_task_insert() with the flag STARPU_HANDLES_SEQUENTIAL_CONSISTENCY

```
char *seq_consistency = malloc(cl.nbuffers * sizeof(char));
seq_consistency[0] = 1;
seq_consistency[1] = 1;
seq_consistency[2] = 0;
ret = starpu_task_insert(&cl,
    STARPU_RW, handleA, STARPU_RW, handleB, STARPU_RW, handleC,
    STARPU_HANDLES_SEQUENTIAL_CONSISTENCY, seq_consistency,
    0);
free(seq_consistency);
```

A full code example is available in the file `examples/dependency/sequential_consistency.c`.
The internal algorithm used by StarPU to set up implicit dependency is as follows:

```
if (sequential_consistency(task) == 1)
    for(i=0 ; i<STARPU_TASK_GET_NBUFFERS(task) ; i++)
      if (sequential_consistency(i-th data, task) == 1)
        if (sequential_consistency(i-th data) == 1)
           create_implicit_dependency(...)
```

33.1.2 Tasks And Tags Dependencies

One can explicitly set dependencies between tasks using starpu_task_declare_deps() or starpu_task_declare_deps_array().
Dependencies between tasks can be expressed through tags associated to a tag with the field starpu_task::tag_id and using the function starpu_tag_declare_deps() or starpu_tag_declare_deps_array(). The example `tests/main/tag_task_data_deps.c` shows how to set dependencies between tasks with different functions.
The termination of a task can be delayed through the function starpu_task_end_dep_add() which specifies the number of calls to the function starpu_task_end_dep_release() needed to trigger the task termination. One can also use starpu_task_declare_end_deps() or starpu_task_declare_end_deps_array() to delay the termination of a task until the termination of other tasks. A simple example is available in the file `tests/main/task_end_dep.c`.
starpu_tag_notify_from_apps() can be used to explicitly unlock a specific tag, but if it is called several times on the

same tag, notification will be done only on first call. However, one can call starpu_tag_restart() to clear the already notified status of a tag which is not associated with a task, and then calling starpu_tag_notify_from_apps() again will notify the successors. Alternatively, starpu_tag_notify_restart_from_apps() can be used to atomically call both starpu_tag_notify_from_apps() and starpu_tag_restart() on a specific tag.
To get the task associated to a specific tag, one can call starpu_tag_get_task(). Once the corresponding task has been executed and when there is no other tag that depend on this tag anymore, one can call starpu_tag_remove() to release the resources associated to the specific tag.

33.2 Waiting For Tasks

StarPU provides several advanced functions to wait for termination of tasks. One can wait for some explicit tasks, or for some tag attached to some tasks, or for some data results.
starpu_task_wait_array() is a function that waits for an array of tasks to complete their execution. starpu_task_wait_for_all_in_ctx() is a function that waits for all tasks in a specific context to complete their execution. starpu_task_wait_for_n_submitted_in_ctx() is a function that waits for a specified number of tasks to be submitted to a specific context. starpu_task_wait_for_no_ready() is a function that waits for all tasks to become unready, which means that they are either completed or blocked on a data dependency. In order to successfully call these functions to wait for termination of tasks, starpu_task::detach should be set to 0 before task submission.
The function starpu_task_nready() returns the number of tasks that are ready to execute, which means that all their data dependencies are satisfied and they are waiting to be scheduled, while the function starpu_task_nsubmitted() returns the number of tasks that have been submitted and not completed yet.
The function starpu_task_finished() can be used to determine whether a specific task has completed its execution.
starpu_tag_wait() and starpu_tag_wait_array() are two blocking functions that can be used to wait for tasks with specific tags to complete their execution. The former one waits for a specified task to complete while the latter one waits for a group of tasks to complete.
When using e.g. starup_task_insert(), it may be more convenient to wait for the *result* of a task rather than waiting for a given task explicitly. That can be done thanks to starpu_data_acquire() or starpu_data_acquire_cb() that wait for the result to be available in the home node of the data. That will thus wait for all the tasks that lead to that result. One can also use starpu_data_acquire_on_node() and give it STARPU_ACQUIRE_NO_NODE to tell to just wait for tasks to complete, but not wait for the data to be available in the home node. One can also use starpu_data_acquire_try() or starpu_data_acquire_on_node_try() to just test for the termination.
If a task is created by using starpu_task_create() or starpu_task_insert(), the field starpu_task::destroy is set to 1 by default, which means that the task structure will be automatically freed after termination. On the other hand, if the task is initialized by using starpu_task_init(), the field starpu_task::destroy is set to 0 by default, which means that the task structure will not be freed until starpu_task_destroy() is called explicitly. Otherwise, we can manually set starpu_task::destroy to 1 before submission or call starpu_task_set_destroy() after submission to activate the automatic freeing of the task structure.

33.3 Using Multiple Implementations Of A Codelet

One may want to write multiple implementations of a codelet for a single type of device and let StarPU choose which one to run. As an example, we will show how to use SSE to scale a vector. The codelet can be written as follows:

```
#include <xmmintrin.h>
void scal_sse_func(void *buffers[], void *cl_arg)
{
    float *vector = (float *) STARPU_VECTOR_GET_PTR(buffers[0]);
    unsigned int n = STARPU_VECTOR_GET_NX(buffers[0]);
    unsigned int n_iterations = n/4;
    if (n % 4 != 0)
        n_iterations++;
    __m128 *VECTOR = (__m128*) vector;
    __m128 factor __attribute__((aligned(16)));
    factor = _mm_set1_ps(*(float *) cl_arg);
    unsigned int i;
    for (i = 0; i < n_iterations; i++)
        VECTOR[i] = _mm_mul_ps(factor, VECTOR[i]);
}
struct starpu_codelet cl =
{
    .cpu_funcs = { scal_cpu_func, scal_sse_func },
    .cpu_funcs_name = { "scal_cpu_func", "scal_sse_func" },
    .nbuffers = 1,
    .modes = { STARPU_RW }
};
```

The full code of this example is available in the file `examples/basic_examples/vector_scal.c`.
Schedulers which are multi-implementation aware (only `dmda` and `pheft` for now) will use the performance models of all the provided implementations, and pick the one which seems to be the fastest.

33.4 Enabling Implementation According To Capabilities

Some implementations may not run on some devices. For instance, some CUDA devices do not support double floating point precision, and thus the kernel execution would just fail; or the device may not have enough shared memory for the implementation being used. The field starpu_codelet::can_execute permits to express this. For instance:

```
static int can_execute(unsigned workerid, struct starpu_task *task, unsigned nimpl)
{
  const struct cudaDeviceProp *props;
  if (starpu_worker_get_type(workerid) == STARPU_CPU_WORKER)
    return 1;
  /* Cuda device */
  props = starpu_cuda_get_device_properties(workerid);
  if (props->major >= 2 || props->minor >= 3)
    /* At least compute capability 1.3, supports doubles */
    return 1;
  /* Old card, does not support doubles */
  return 0;
}
struct starpu_codelet cl =
{
    .can_execute = can_execute,
    .cpu_funcs = { cpu_func },
    .cpu_funcs_name = { "cpu_func" },
    .cuda_funcs = { gpu_func }
    .nbuffers = 1,
    .modes = { STARPU_RW }
};
```

A full example is available in the file `examples/reductions/dot_product.c`.
This can be essential e.g. when running on a machine which mixes various models of CUDA devices, to take benefit from the new models without crashing on old models.
Note: the function starpu_codelet::can_execute is called by the scheduler each time it tries to match a task with a worker, and should thus be very fast. The function starpu_cuda_get_device_properties() provides quick access to CUDA properties of CUDA devices to achieve such efficiency.
Another example is to compile CUDA code for various compute capabilities, resulting with two CUDA functions, e.g. `scal_gpu_13` for compute capability 1.3, and `scal_gpu_20` for compute capability 2.0. Both functions can be provided to StarPU by using starpu_codelet::cuda_funcs, and starpu_codelet::can_execute can then be used to rule out the `scal_gpu_20` variant on a CUDA device which will not be able to execute it:

```
static int can_execute(unsigned workerid, struct starpu_task *task, unsigned nimpl)
{
  const struct cudaDeviceProp *props;
  if (starpu_worker_get_type(workerid) == STARPU_CPU_WORKER)
    return 1;
  /* Cuda device */
  if (nimpl == 0)
    /* Trying to execute the 1.3 capability variant, we assume it is ok in all cases.    */
    return 1;
  /* Trying to execute the 2.0 capability variant, check that the card can do it.    */
  props = starpu_cuda_get_device_properties(workerid);
  if (props->major >= 2 || props->minor >= 0)
    /* At least compute capability 2.0, can run it */
    return 1;
  /* Old card, does not support 2.0, will not be able to execute the 2.0 variant.    */
  return 0;
}
struct starpu_codelet cl =
{
    .can_execute = can_execute,
    .cpu_funcs = { cpu_func },
    .cpu_funcs_name = { "cpu_func" },
    .cuda_funcs = { scal_gpu_13, scal_gpu_20 },
    .nbuffers = 1,
    .modes = { STARPU_RW }
};
```

Another example is having specialized implementations for some given common sizes, for instance here we have a specialized implementation for 1024x1024 matrices:

```
static int can_execute(unsigned workerid, struct starpu_task *task, unsigned nimpl)
{
  const struct cudaDeviceProp *props;
  if (starpu_worker_get_type(workerid) == STARPU_CPU_WORKER)
    return 1;
  /* Cuda device */
  switch (nimpl)
  {
    case 0:
      /* Trying to execute the generic capability variant.    */
      return 1;
    case 1:
    {
      /* Trying to execute the size == 1024 specific variant.    */
      struct starpu_matrix_interface *interface = starpu_data_get_interface_on_node(task->handles[0]);
      return STARPU_MATRIX_GET_NX(interface) == 1024 && STARPU_MATRIX_GET_NY(interface == 1024);
    }
  }
}
struct starpu_codelet cl =
{
    .can_execute = can_execute,
    .cpu_funcs = { cpu_func },
    .cpu_funcs_name = { "cpu_func" },
    .cuda_funcs = { potrf_gpu_generic, potrf_gpu_1024 },
    .nbuffers = 1,
    .modes = { STARPU_RW }
};
```

Note that the most generic variant should be provided first, as some schedulers are not able to try the different variants.

33.5 Getting Task Children

It may be interesting to get the list of tasks which depend on a given task, notably when using implicit dependencies, since this list is computed by StarPU. starpu_task_get_task_succs() or starpu_task_get_task_scheduled_succs() provides it. For instance:

```
struct starpu_task *tasks[4];
ret = starpu_task_get_task_succs(task, sizeof(tasks)/sizeof(*tasks), tasks);
```

And the full example of getting task children is available in the file `tests/main/get_children_tasks.c`

33.6 Parallel Tasks

StarPU can leverage existing parallel computation libraries by the means of parallel tasks. A parallel task is a task which is run by a set of CPUs (called a parallel or combined worker) at the same time, by using an existing parallel CPU implementation of the computation to be achieved. This can also be useful to improve the load balance between slow CPUs and fast GPUs: since CPUs work collectively on a single task, the completion time of tasks on CPUs become comparable to the completion time on GPUs, thus relieving from granularity discrepancy concerns. `hwloc` support needs to be enabled to get good performance, otherwise StarPU will not know how to better group cores.

Two modes of execution exist to accommodate with existing usages.

33.6.1 Fork-mode Parallel Tasks

In the Fork mode, StarPU will call the codelet function on one of the CPUs of the combined worker. The codelet function can use starpu_combined_worker_get_size() to get the number of threads it is allowed to start to achieve the computation. The CPU binding mask for the whole set of CPUs is already enforced, so that threads created by the function will inherit the mask, and thus execute where StarPU expected, the OS being in charge of choosing how to schedule threads on the corresponding CPUs. The application can also choose to bind threads by hand, using e.g. `sched_getaffinity` to know the CPU binding mask that StarPU chose.

For instance, using OpenMP (full source is available in `examples/openmp/vector_scal.c`):

```
void scal_cpu_func(void *buffers[], void *_args)
{
    unsigned i;
    float *factor = _args;
    struct starpu_vector_interface *vector = buffers[0];
    unsigned n = STARPU_VECTOR_GET_NX(vector);
    float *val = (float *)STARPU_VECTOR_GET_PTR(vector);
#pragma omp parallel for num_threads(starpu_combined_worker_get_size())
    for (i = 0; i < n; i++)
        val[i] *= *factor;
}
static struct starpu_codelet cl =
{
    .modes = { STARPU_RW },
    .where = STARPU_CPU,
    .type = STARPU_FORKJOIN,
    .max_parallelism = INT_MAX,
    .cpu_funcs = {scal_cpu_func},
    .cpu_funcs_name = {"scal_cpu_func"},
    .nbuffers = 1,
};
```

Other examples include for instance calling a BLAS parallel CPU implementation (see `examples/mult/xgemm.c`).

33.6.2 SPMD-mode Parallel Tasks

In the SPMD mode, StarPU will call the codelet function on each CPU of the combined worker. The codelet function can use starpu_combined_worker_get_size() to get the total number of CPUs involved in the combined worker, and thus the number of calls that are made in parallel to the function, and starpu_combined_worker_get_rank() to get the rank of the current CPU within the combined worker. For instance:

```
static void func(void *buffers[], void *args)
{
    unsigned i;
    float *factor = _args;
    struct starpu_vector_interface *vector = buffers[0];
    unsigned n = STARPU_VECTOR_GET_NX(vector);
    float *val = (float *)STARPU_VECTOR_GET_PTR(vector);
    /* Compute slice to compute */
    unsigned m = starpu_combined_worker_get_size();
    unsigned j = starpu_combined_worker_get_rank();
    unsigned slice = (n+m-1)/m;
    for (i = j * slice; i < (j+1) * slice && i < n; i++)
        val[i] *= *factor;
}
static struct starpu_codelet cl =
{
    .modes = { STARPU_RW },
    .type = STARPU_SPMD,
    .max_parallelism = INT_MAX,
    .cpu_funcs = { func },
    .cpu_funcs_name = { "func" },
    .nbuffers = 1,
}
```

A full example is available in `examples/spmd/vector_scal_spmd.c`.

Of course, this trivial example will not really benefit from parallel task execution, and was only meant to be simple to understand. The benefit comes when the computation to be done is so that threads have to e.g. exchange intermediate results, or write to the data in a complex but safe way in the same buffer.

33.6.3 Parallel Tasks Performance

To benefit from parallel tasks, a parallel-task-aware StarPU scheduler has to be used. When exposed to codelets with a flag STARPU_FORKJOIN or STARPU_SPMD, the schedulers `pheft` (parallel-heft) and `peager` (parallel eager) will indeed also try to execute tasks with several CPUs. It will automatically try the various available combined worker sizes (making several measurements for each worker size) and thus be able to avoid choosing a large combined worker if the codelet does not actually scale so much. Examples using parallel-task-aware StarPU scheduler are available in `tests/parallel_tasks/parallel_kernels.c` and `tests/parallel_tasks/parallel_kernels_spmd.c`.

This is however for now only proof of concept, and has not really been optimized yet.

33.6.4 Combined Workers

By default, StarPU creates combined workers according to the architecture structure as detected by `hwloc`. It means that for each object of the `hwloc` topology (NUMA node, socket, cache, ...) a combined worker will be created. If some nodes of the hierarchy have a big arity (e.g. many cores in a socket without a hierarchy of shared caches), StarPU will create combined workers of intermediate sizes. The variable STARPU_SYNTHESIZE_ARITY_COMBINED_WORKER permits to tune the maximum arity between levels of combined workers.

The combined workers actually produced can be seen in the output of the tool `starpu_machine_display` (the environment variable STARPU_SCHED has to be set to a combined worker-aware scheduler such as `pheft` or `peager`).

33.6.5 Concurrent Parallel Tasks

Unfortunately, many environments and libraries do not support concurrent calls.
For instance, most OpenMP implementations (including the main ones) do not support concurrent `pragma omp parallel` statements without nesting them in another `pragma omp parallel` statement, but StarPU does not yet support creating its CPU workers by using such pragma.
Other parallel libraries are also not safe when being invoked concurrently from different threads, due to the use of global variables in their sequential sections, for instance.
The solution is then to use only one combined worker at a time. This can be done by setting the field starpu_conf::single_combined_worker to `1`, or setting the environment variable STARPU_SINGLE_COMBINED_WORKER to `1`. StarPU will then run only one parallel task at a time (but other CPU and GPU tasks are not affected and can be run concurrently). The parallel task scheduler will however still try varying combined worker sizes to look for the most efficient ones. A full example is available in `examples/spmd/vector_scal_spmd.c`.

33.7 Synchronization Tasks

For the application convenience, it may be useful to define tasks which do not actually make any computation, but wear for instance dependencies between other tasks or tags, or to be submitted in callbacks, etc.
The obvious way is of course to make kernel functions empty, but such task will thus have to wait for a worker to become ready, transfer data, etc.
A much lighter way to define a synchronization task is to set its field starpu_task::cl to `NULL`. The task will thus be a mere synchronization point, without any data access or execution content: as soon as its dependencies become available, it will terminate, call the callbacks, and release dependencies.
An intermediate solution is to define a codelet with its field starpu_codelet::where set to STARPU_NOWHERE, for instance:

```
struct starpu_codelet cl =
{
    .where = STARPU_NOWHERE,
    .nbuffers = 1,
    .modes = { STARPU_R },
}
task = starpu_task_create();
task->cl = &cl;
task->handles[0] = handle;
starpu_task_submit(task);
```

will create a task which simply waits for the value of `handle` to be available for read. This task can then be depended on, etc. A full example is available in `examples/filters/fmultiple_manual.c`.
StarPU provides starpu_task_create_sync() to create a new synchronization task, the same as the previous example but without submitting the task. The function starpu_create_sync_task() is also used to create a new synchronization task and submit it, which is a task that waits for specific tags and calls the specified callback function when the task is finished. The function starpu_create_callback_task() can create and submit a synchronization task, which is a task that completes immediately and calls the specified callback function right after.

Chapter 34

Advanced Data Management

34.1 Data Interface with Variable Size

Besides the data interfaces already available in StarPU, mentioned in Data Interface, tasks are actually allowed to change the size of data interfaces.

The simplest case is just changing the amount of data actually used within the allocated buffer. This is for instance implemented for the matrix interface: one can set the new NX/NY values with STARPU_MATRIX_SET_NX(), STARPU_MATRIX_SET_NY(), and STARPU_MATRIX_SET_LD() at the end of the task implementation. Data transfers achieved by StarPU will then use these values instead of the whole allocated size. The values of course need to be set within the original allocation. To reserve room for increasing the NX/NY values, one can use starpu_matrix_data_register_allocsize() instead of starpu_matrix_data_register(), to specify the allocation size to be used instead of the default NX∗NY∗ELEMSIZE. It is also available for a vector by using starpu_vector_data_register_allocsize() to specify the allocation size to be used instead of the default NX∗ELEMSIZE. To support this, the data interface has to implement the functions starpu_data_interface_ops::alloc_footprint, starpu_data_interface_ops::alloc_compare, and starpu_data_interface_ops::reuse_data_ for proper StarPU allocation management. It might be useful to implement starpu_data_interface_ops::cache_data_on_node, otherwise StarPU will just call `memcpy()`.

A more involved case is changing the amount of allocated data. The task implementation can just reallocate the buffer during its execution, and set the proper new values in the interface structure, e.g. nx, ny, ld, etc. so that the StarPU core knows the new data layout. The structure starpu_data_interface_ops however then needs to have the field starpu_data_interface_ops::dontcache set to 1, to prevent StarPU from trying to perform any cached allocation, since the allocated size will vary. An example is available in `tests/datawizard/variable_size.c`. The example uses its own data interface to contain some simulation information for data growth, but the principle can be applied for any data interface.

The principle is to use starpu_malloc_on_node_flags() to make the new allocation, and use starpu_free_on_node_flags() to release any previous allocation. The flags have to be precisely like in the example:

```
unsigned workerid = starpu_worker_get_id_check();
unsigned dst_node = starpu_worker_get_memory_node(workerid);
interface->ptr = starpu_malloc_on_node_flags(dst_node, size + increase, STARPU_MALLOC_PINNED |
      STARPU_MALLOC_COUNT | STARPU_MEMORY_OVERFLOW);
starpu_free_on_node_flags(dst_node, old, size, STARPU_MALLOC_PINNED | STARPU_MALLOC_COUNT |
      STARPU_MEMORY_OVERFLOW);
interface->size += increase;
```

so that the allocated area has the expected properties and the allocation is properly accounted for.

Depending on the interface (vector, CSR, etc.) you may have to fix several fields of the data interface: e.g. both `nx` and `allocsize` for vectors, and store the pointer both in `ptr` and `dev_handle`.

Some interfaces make a distinction between the actual number of elements stored in the data and the actually allocated buffer. For instance, the vector interface uses the `nx` field for the former, and the `allocsize` for the latter. This allows for lazy reallocation to avoid reallocating the buffer every time to exactly match the actual number of elements. Computations and data transfers will use the field `nx`, while allocation functions will use the field `allocsize`. One just has to make sure that `allocsize` is always bigger or equal to `nx`.

Important note: one can not change the size of a partitioned data.

34.2 Data Management Allocation

When the application allocates data, whenever possible it should use the function starpu_malloc(), which will ask CUDA or OpenCL to make the allocation itself and pin the corresponding allocated memory (a basic example is in `examples/basic_examples/block.c`), or to use the function starpu_memory_pin() to pin memory allocated by other ways, such as local arrays (a basic example is in `examples/basic_examples/vector_scal.c`). This is needed to permit asynchronous data transfer, i.e. permit data transfer to overlap with computations. Otherwise, the trace will show that the state `DriverCopyAsync` takes a lot of time, this is because CUDA or OpenCL then reverts to synchronous transfers. Before shutting down StarPU, the application should deallocate any memory that has previously been allocated with starpu_malloc(), by calling either starpu_free() or starpu_free_noflag() which is more recommended. If the application has pinned memory using starpu_memory_pin(), it should unpin the memory using starpu_memory_unpin() before freeing the memory.

If an application requires a specific alignment constraint for memory allocations made with starpu_malloc(), it can use the starpu_malloc_set_align() function to set the alignment requirement.

The application can provide its own allocation function by calling starpu_malloc_set_hooks(). StarPU will then use them for all data handle allocations in the main memory. An example is in `examples/basic_examples/hooks.c`.

StarPU provides several functions to monitor the memory usage and availability on the system. The application can use the starpu_memory_get_used() function to monitor its own memory usage on a node, and the starpu_memory_get_total_all_nodes() function to monitor the amount of total memory on all memory nodes, and the starpu_memory_get_available_all_nodes() function to monitor the amount of available memory on all memory nodes. Additionally, the starpu_memory_get_used_all_nodes() function can be used to monitor the amount of used memory on all memory nodes.

By default, StarPU leaves replicates of data wherever they were used, in case they will be re-used by other tasks, thus saving the data transfer time. When some task modifies some data, all the other replicates are invalidated, and only the processing unit which ran this task will have a valid replicate of the data. If the application knows that this data will not be re-used by further tasks, it should advise StarPU to immediately replicate it to a desired list of memory nodes (given through a bitmask). This can be understood like the write-through mode of CPU caches.

```
starpu_data_set_wt_mask(img_handle, 1«0);
```

will for instance request to always automatically transfer a replicate into the main memory (node `0`), as bit `0` of the write-through bitmask is being set. An example is available in `examples/pi/pi.c`.

```
starpu_data_set_wt_mask(img_handle, ~0U);
```

will request to always automatically broadcast the updated data to all memory nodes. An example is available in `tests/datawizard/wt_broadcast.c`.

Setting the write-through mask to ∼`0U` can also be useful to make sure all memory nodes always have a copy of the data, so that it is never evicted when memory gets scarce.

Implicit data dependency computation can become expensive if a lot of tasks access the same piece of data. If no dependency is required on some piece of data (e.g. because it is only accessed in read-only mode, or because write accesses are actually commutative), use the function starpu_data_set_sequential_consistency_flag() to disable implicit dependencies on this data.

In the same vein, accumulation of results in the same data can become a bottleneck. The use of the mode STARPU_REDUX permits to optimize such accumulation (see Data Reduction). To a lesser extent, the use of the flag STARPU_COMMUTE keeps the bottleneck (see Commute Data Access), but at least permits the accumulation to happen in any order.

Applications often need a data just for temporary results. In such a case, registration can be made without an initial value, for instance this produces a vector data:

```
starpu_vector_data_register(&handle, -1, 0, n, sizeof(float));
```

StarPU will then allocate the actual buffer only when it is actually needed, e.g. directly on the GPU without allocating in main memory.

In the same vein, once the temporary results are not useful anymore, the data should be thrown away. If the handle is not to be reused, it can be unregistered:

```
starpu_data_unregister_submit(handle);
```

actual unregistration will be done after all tasks working on the handle terminate.

One can also unregister the data handle by calling:

```
starpu_data_unregister_no_coherency(handle);
```

Different from starpu_data_unregister(), a valid copy of the data is not put back into the home node in the buffer that was initially registered.

If the handle is to be reused, instead of unregistering it, it can simply be deinitialized:

```
starpu_data_deinitialize(handle);
```

So that the value will be ignored and not written back to main memory.

Or instead it can even be invalidated (the buffers containing the current value will then be freed, and reallocated only when another task writes some value to the handle):

```
starpu_data_invalidate(handle);
```

if the data transfer is asynchronous, one can use the submit versions:

```
starpu_data_deinitialize_submit(handle);
```

or

```
starpu_data_invalidate_submit(handle);
```

A basic example is available in the files `tests/datawizard/data_deinitialize.c` and `tests/datawizard/data_invalidation.c`.

34.3 Data Access

To access registered data outside tasks we can call the function starpu_data_acquire(). The access mode can be read-only mode STARPU_R, write-only mode STARPU_W, and read-write mode STARPU_RW. We will get an up-to-date copy of handle in memory located where the data was originally registered. The application can also call starpu_data_acquire_try() instead of starpu_data_acquire() to acquire the data, but if previously-submitted tasks have not completed when we ask to acquire the data, the program will crash. starpu_data_release() must be called once the application no longer needs to access the piece of data. Or call starpu_data_release_to() to partly release the piece of data acquired. We can also access registered data from a given memory node by calling the function starpu_data_acquire_on_node(), or calling starpu_data_acquire_on_node_try() if all previously-submitted tasks have completed. Correspondingly, starpu_data_release_on_node() must be called once the application no longer needs to access the piece of data and the node parameter must be exactly the same as the corresponding starpu_data_acquire_on_node() call. Or call starpu_data_release_to_on_node() to partly release the piece of data acquired.

The application may access the requested data asynchronous during the execution of callback by calling starpu_data_acquire_cb(), and by calling starpu_data_acquire_cb_sequential_consistency() with the possibility of enabling or disabling data dependencies. The callback function must call starpu_data_release() once the application no longer needs to access the piece of data. Or call starpu_data_release_to() to partly release the piece of data acquired. The application can also access registered data from a given memory node instead of main memory by calling the function starpu_data_acquire_on_node_cb(), and by calling starpu_data_acquire_on_node_cb_sequential_consistency() with the possibility of enabling or disabling data dependencies. starpu_data_release_on_node() must be called once the application no longer needs to access the piece of data. Or call starpu_data_release_to_on_node() to partly release the piece of data acquired.

34.4 Data Prefetch

The scheduling policies `heft`, `dmda` and `pheft` perform data prefetch (see STARPU_PREFETCH): as soon as a scheduling decision is taken for a task, requests are issued to transfer its required data to the target processing unit, if needed, so that when the processing unit actually starts the task, its data will hopefully be already available, and it will not have to wait for the transfer to finish.

The application may want to perform some manual prefetching, for several reasons such as excluding initial data transfers from performance measurements, or setting up an initial statically-computed data distribution on the machine before submitting tasks, which will thus guide StarPU toward an initial task distribution (since StarPU will try to avoid further transfers).

This can be achieved by giving the function starpu_data_prefetch_on_node() the handle and the desired target memory node. An example is available in the file `tests/microbenchs/prefetch_data_on_node.c`. The variant starpu_data_idle_prefetch_on_node() can be used to issue the transfer only when the bus is idle. One can also call starpu_data_request_allocation() for the allocation of a piece of data on the specified memory node. We can know whether the allocation is done on the specified memory node by using starpu_data_test_if_allocated_on_node(). We can also know whether the map is done on the specified memory node by using starpu_data_test_if_mapped_on_node().

If we want higher priority to request data to be replicated to a given node as soon as possible, so that it is available there for tasks, we can call starpu_data_fetch_on_node(). We can call starpu_data_prefetch_on_node_prio() to have a priority than starpu_data_prefetch_on_node(). And call starpu_data_idle_prefetch_on_node_prio() to have a bit higher priority than starpu_data_idle_prefetch_on_node().

Conversely, one can advise StarPU that some data will not be useful in the close future by calling starpu_data_wont_use(). StarPU will then write its value back to its home node, and evict it from GPUs when room is needed. An example is available in the file `tests/datawizard/partition_wontuse.c`. One can

also advise StarPU to evict data from the memory node directly by calling starpu_data_evict_from_node(), but it may fail if e.g. some tasks are still working on the memory node. To avoid failure one can call starpu_data_can_evict() to check whether data can be evicted from the memory node. Anyway it is more recommended to use starpu_data_wont_use().
One can query the status of handle on the specified memory node by calling starpu_data_query_status2() or starpu_data_query_status(). One can call starpu_memchunk_tidy() to tidy the available memory on the specified memory node periodically.

34.5 Manual Partitioning

Except the partitioning functions described in Partitioning Data and Asynchronous Partitioning, one can also handle partitioning by hand, by registering several views on the same piece of data. The idea is then to manage the coherency of the various views through the common buffer in the main memory. `examples/filters/fmultiple_manual.c` is a complete example using this technique.
In short, we first register the same matrix several times:

```
starpu_matrix_data_register(&handle, STARPU_MAIN_RAM, (uintptr_t)matrix, NX, NX, NY, sizeof(matrix[0]));
for (i = 0; i < PARTS; i++)
    starpu_matrix_data_register(&vert_handle[i], STARPU_MAIN_RAM, (uintptr_t)&matrix[0][i*(NX/PARTS)], NX,
      NX/PARTS, NY, sizeof(matrix[0][0]));
```

Since StarPU is not aware that the two handles are actually pointing to the same data, we have a danger of inadvertently submitting tasks to both views, which will bring a mess since StarPU will not guarantee any coherency between the two views. To make sure we don't do this, we invalidate the view that we will not use:

```
for (i = 0; i < PARTS; i++)
    starpu_data_invalidate(vert_handle[i]);
```

Then we can safely work on `handle`.
When we want to switch to the vertical slice view, all we need to do is bring coherency between them by running an empty task on the home node of the data:

```
struct starpu_codelet cl_switch =
{
    .where = STARPU_NOWHERE,
    .nbuffers = 3,
    .specific_nodes = 1,
    .nodes = { STARPU_MAIN_RAM, STARPU_MAIN_RAM, STARPU_MAIN_RAM },
};
ret = starpu_task_insert(&cl_switch, STARPU_RW, handle,
            STARPU_W, vert_handle[0],
            STARPU_W, vert_handle[1],
            0);
```

The execution of the task `switch` will get back the matrix data into the main memory, and thus the vertical slices will get the updated value there.
Again, we prefer to make sure that we don't accidentally access the matrix through the whole-matrix handle:

```
starpu_data_invalidate_submit(handle);
```

Note: when enabling a set of handles in this way, the set must not have any overlapping, i.e. the handles of the set must not have any part of data in common, otherwise StarPU will not properly handle concurrent accesses between them.
And now we can start using vertical slices, etc.

34.6 Data handles helpers

Functions starpu_data_set_user_data() and starpu_data_get_user_data() are used to associate user-defined data with a specific data handle. One can set or retrieve the field `user_data` of the data handle by calling these two functions respectively. Similarly, functions starpu_data_set_sched_data() and starpu_data_get_sched_data() are used to associate scheduling-related data with a specific data handle. One can set or retrieve the field `sched_data` of the data handle by calling these two functions respectively. One can set a name for a data handle by calling starpu_data_set_name().
One can call starpu_data_register_same() to register a new piece of data into a data handle with the same interface as the specified data handle. If necessary, one can register a void interface by using starpu_void_data_register(). There is no data really associated to this interface, but it may be used as a synchronization mechanism.
One can call starpu_data_cpy() or starpu_data_cpy_priority() to copy data from one memory location to another memory location, but the latter one allows the application to specify a priority value for the copy operation. The higher the priority value, the sonner the copy operation will be scheduled and executed. One can also call starpu_data_dup_ro() function for duplicating, but this function only creates a new read-only data block that is an exact copy of the original data block. The new data block can be used independently of the original data block

for read-only access.
starpu_data_pack_node() and starpu_data_pack() are functions that are used to pack a data item into a binary buffer on a node or on local memory node. starpu_data_peek_node() and starpu_data_peek() are functions that allow you to read in handle's node or local node replicate the data located at the given pointer. starpu_data_unpack_node() and starpu_data_unpack() are functions that are used to unpack a data item from a binary buffer on a node or on local memory node.
StarPU provides several functions for querying the size and memory allocation of variable size data items, such as: starpu_data_get_size() is a function that returns the size of a data associated with handle in bytes. This is the size of the actual data stored in memory. starpu_data_get_alloc_size() is a function that returns the amount of memory that has been allocated for a data associated with handle in anticipation. This may be larger than the actual size of the data item, due to alignment requirements or other implementation details. starpu_data_get_max_size() is a function that returns the maximum size of a handle data that can be allocated by StarPU.
One can call starpu_data_get_home_node() to retrieve the identifier of the node on which the data handle is originally stored. One can call starpu_data_print() to print basic information about the data handle and the node to the specified file.

34.7 Handles data buffer pointers

A simple understanding of StarPU handles is that it's a collection of buffers on each memory node of the machine, which contain the same data. The picture is however made more complex with the OpenCL support and with partitioning.
When partitioning a handle, the data buffers of the subhandles will indeed be inside the data buffers of the main handle (to save transferring data back and forth between the main handle and the subhandles). But in OpenCL, a `cl_mem` is not a pointer, but an opaque value on which pointer arithmetic can not be used. That is why data interfaces contain three fields: `dev_handle`, `offset`, and `ptr`.

- The field `dev_handle` is what the allocation function returned, and one can not do arithmetic on it.
- The field `offset` is the offset inside the allocated area, most often it will be 0 because data start at the beginning of the allocated area, but when the handle is partitioned, the subhandles will have varying `offset` values, for each subpiece.
- The field `ptr`, in the non-OpenCL case, i.e. when pointer arithmetic can be used on `dev_handle`, is just the sum of `dev_handle` and `offset`, provided for convenience.

This means that:

- computation kernels can use `ptr` in non-OpenCL implementations.
- computation kernels have to use `dev_handle` and `offset` in the OpenCL implementation.
- allocation methods of data interfaces have to store the value returned by starpu_malloc_on_node() in `dev_handle` and `ptr`, and set `offset` to 0.
- partitioning filters have to copy over `dev_handle` without modifying it, set in the child different values of `offset`, and set `ptr` accordingly as the sum of `dev_handle` and `offset`.

We can call starpu_data_handle_to_pointer() to get `ptr` associated with the data handle, or call starpu_data_get_local_ptr() to get the local pointer associated with the data handle.
Examples in the directory `examples/interface/complex_dev_handle/` show how to generate and implement an interface supporting OpenCL.
To better notice the difference between simple `ptr` and `dev_handle + offset`, one can compare `examples/interface/complex_interface.c` vs `examples/interface/complex_dev_handle/complex_dev_handle_interface.c` and `examples/interface/complex_filters.c` vs `examples/interface/complex_dev_handle/complex_dev_handle_filters.c`.

34.8 Defining A New Data Filter

StarPU provides a series of predefined filters in Data Partition, but additional filters can be defined by the application. The principle is that the filter function just fills the memory location of the `i-th` subpart of a data. Examples are provided in `src/datawizard/interfaces/*_filters.c`, check starpu_data_filter::filter_func for further details. The helper function starpu_filter_nparts_compute_chunk_size_and_offset() can be used to compute the division of pieces of data.

34.9 Defining A New Data Interface

This section proposes an example how to define your own interface, when the StarPU-provided interface do not fit your needs. Here we take a simple example of an array of complex numbers represented by two arrays of double values. The full source code is in `examples/interface/complex_interface.c` and `examples/interface/complex_interface.h`

Let's thus define a new data interface to manage arrays of complex numbers:

```
/* interface for complex numbers */
struct starpu_complex_interface
{
        double *real;
        double *imaginary;
        int nx;
};
```

That structure stores enough to describe **one** buffer of such kind of data. It is used for the buffer stored in the main memory, another instance is used for the buffer stored in a GPU, etc. A *data handle* is thus a collection of such structures, to describe each buffer on each memory node.

Note: one should not make pointers that point into such structures, because StarPU needs to be able to copy over the content of it to various places, for instance to efficiently migrate a data buffer from one data handle to another data handle, so the actual address of the structure may vary.

34.9.1 Data registration

Registering such a data to StarPU is easily done using the function starpu_data_register(). The last parameter of the function, `interface_complex_ops`, will be described below.

```
void starpu_complex_data_register(starpu_data_handle_t *handleptr,
     unsigned home_node, double *real, double *imaginary, int nx)
{
        struct starpu_complex_interface complex =
        {
                .real = real,
                .imaginary = imaginary,
                .nx = nx
        };
        starpu_data_register(handleptr, home_node, &complex, &interface_complex_ops);
}
```

The `struct starpu_complex_interface complex` is here used just to store the parameters provided by users to `starpu_complex_data_register`. starpu_data_register() will first allocate the handle, and then pass the structure `starpu_complex_interface` to the method starpu_data_interface_ops::register_data_handle, which records them within the data handle (it is called once per node by starpu_data_register()):

```
static void complex_register_data_handle(starpu_data_handle_t handle, int home_node, void *data_interface)
{
    struct starpu_complex_interface *complex_interface = (struct starpu_complex_interface *) data_interface;
    unsigned node;
    for (node = 0; node < STARPU_MAXNODES; node++)
    {
        struct starpu_complex_interface *local_interface = (struct starpu_complex_interface *)
            starpu_data_get_interface_on_node(handle, node);
        local_interface->nx = complex_interface->nx;
        if (node == home_node)
        {
            local_interface->real = complex_interface->real;
            local_interface->imaginary = complex_interface->imaginary;
        }
        else
        {
            local_interface->real = NULL;
            local_interface->imaginary = NULL;
        }
    }
}
```

If the application provided a home node, the corresponding pointers will be recorded for that node. Others have no buffer allocated yet. Possibly the interface needs some dynamic allocation (e.g. to store an array of dimensions that can have variable size). The corresponding deallocation will then be done in starpu_data_interface_ops::unregister_data_handle.

Different operations need to be defined for a data interface through the type starpu_data_interface_ops. We only define here the basic operations needed to run simple applications. The source code for the different functions can be found in the file `examples/interface/complex_interface.c`, the details of the hooks to be provided are documented in starpu_data_interface_ops .

```
static struct starpu_data_interface_ops interface_complex_ops =
{
        .register_data_handle = complex_register_data_handle,
        .allocate_data_on_node = complex_allocate_data_on_node,
        .copy_methods = &complex_copy_methods,
        .get_size = complex_get_size,
        .footprint = complex_footprint,
        .interfaceid = STARPU_UNKNOWN_INTERFACE_ID,
        .interface_size = sizeof(struct starpu_complex_interface),
};
```

The field starpu_data_interface_ops::interfaceid should be defined to STARPU_UNKNOWN_INTERFACE_ID when defining the interface, its value will be updated the first time a data is registered through the new data interface.

Convenience functions can be defined to access the different fields of the complex interface from a StarPU data handle after a call to starpu_data_acquire():

```
double *starpu_complex_get_real(starpu_data_handle_t handle)
{
        struct starpu_complex_interface *complex_interface =
          (struct starpu_complex_interface *) starpu_data_get_interface_on_node(handle, STARPU_MAIN_RAM);
        return complex_interface->real;
}
double *starpu_complex_get_imaginary(starpu_data_handle_t handle);
int starpu_complex_get_nx(starpu_data_handle_t handle);
```

Similar functions need to be defined to access the different fields of the complex interface from a `void *` pointer to be used within codelet implementations.

```
#define STARPU_COMPLEX_GET_REAL(interface)      (((struct starpu_complex_interface *)(interface))->real)
#define STARPU_COMPLEX_GET_IMAGINARY(interface) (((struct starpu_complex_interface
      *)(interface))->imaginary)
#define STARPU_COMPLEX_GET_NX(interface)        (((struct starpu_complex_interface *)(interface))->nx)
```

Complex data interfaces can then be registered to StarPU.

```
double real = 45.0;
double imaginary = 12.0;
starpu_complex_data_register(&handle1, STARPU_MAIN_RAM, &real, &imaginary, 1);
starpu_task_insert(&cl_display, STARPU_R, handle1, 0);
```

and used by codelets.

```
void display_complex_codelet(void *descr[], void *_args)
{
        int nx = STARPU_COMPLEX_GET_NX(descr[0]);
        double *real = STARPU_COMPLEX_GET_REAL(descr[0]);
        double *imaginary = STARPU_COMPLEX_GET_IMAGINARY(descr[0]);
        int i;
        for(i=0 ; i<nx ; i++)
        {
                fprintf(stderr, "Complex[%d] = %3.2f + %3.2f i\n", i, real[i], imaginary[i]);
        }
}
```

The whole code for this complex data interface is available in the directory `examples/interface/`.

34.9.2 Data footprint

We need to pass a custom footprint function to the method starpu_data_interface_ops::footprint which computes data size footprint. StarPU provides several functions to compute different type of value : starpu_hash_crc32c_be_n() is used to compute the CRC of a byte buffer, starpu_hash_crc32c_be_ptr() is used to compute the CRC of a pointer value, starpu_hash_crc32c_be() is used to compute the CRC of a 32bit number, starpu_hash_crc32c_string() is used to compute the CRC of a string.

34.9.3 Data allocation

To be able to run tasks on GPUs etc. StarPU needs to know how to allocate a buffer for the interface. In our example, two allocations are needed in the allocation method `complex_allocate_data_on_node()`: one for the real part and one for the imaginary part.

```
static starpu_ssize_t complex_allocate_data_on_node(void *data_interface, unsigned node)
{
    struct starpu_complex_interface *complex_interface = (struct starpu_complex_interface *) data_interface;
    double *addr_real = NULL;
    double *addr_imaginary = NULL;
    starpu_ssize_t requested_memory = complex_interface->nx * sizeof(complex_interface->real[0]);
    addr_real = (double*) starpu_malloc_on_node(node, requested_memory);
    if (!addr_real)
        goto fail_real;
    addr_imaginary = (double*) starpu_malloc_on_node(node, requested_memory);
    if (!addr_imaginary)
        goto fail_imaginary;
    /* update the data properly in consequence */
    complex_interface->real = addr_real;
    complex_interface->imaginary = addr_imaginary;
    return 2*requested_memory;
fail_imaginary:
    starpu_free_on_node(node, (uintptr_t) addr_real, requested_memory);
fail_real:
    return -ENOMEM;
}
```

Here we try to allocate the two parts. If either of them fails, we return `-ENOMEM`. If they succeed, we can record the obtained pointers and returned the amount of allocated memory (for memory usage accounting).

Conversely, `complex_free_data_on_node()` frees the two parts:

```
static void complex_free_data_on_node(void *data_interface, unsigned node)
{
    struct starpu_complex_interface *complex_interface = (struct starpu_complex_interface *) data_interface;
    starpu_ssize_t requested_memory = complex_interface->nx * sizeof(complex_interface->real[0]);
    starpu_free_on_node(node, (uintptr_t) complex_interface->real, requested_memory);
    starpu_free_on_node(node, (uintptr_t) complex_interface->imaginary, requested_memory);
}
```

We can call starpu_opencl_allocate_memory() to allocate memory on an OpenCL device.

We have not made anything particular for GPUs or whatsoever: it is starpu_free_on_node() which knows how to actually make the allocation, and returns the resulting pointer, be it in main memory, in GPU memory, etc.

34.9.4 Data copy

Now that StarPU knows how to allocate/free a buffer, it needs to be able to copy over data into/from it. Defining a method `copy_any_to_any()` allows StarPU to perform direct transfers between main memory and GPU memory.

```
static int copy_any_to_any(void *src_interface, unsigned src_node,
               void *dst_interface, unsigned dst_node,
               void *async_data)
{
    struct starpu_complex_interface *src_complex = src_interface;
    struct starpu_complex_interface *dst_complex = dst_interface;
    int ret = 0;
    if (starpu_interface_copy((uintptr_t) src_complex->real, 0, src_node,
                    (uintptr_t) dst_complex->real, 0, dst_node,
                     src_complex->nx*sizeof(src_complex->real[0]),
                     async_data))
        ret = -EAGAIN;
    if (starpu_interface_copy((uintptr_t) src_complex->imaginary, 0, src_node,
                    (uintptr_t) dst_complex->imaginary, 0, dst_node,
                     src_complex->nx*sizeof(src_complex->imaginary[0]),
                     async_data))
        ret = -EAGAIN;
    return ret;
}
```

We here again have no idea what is main memory or GPU memory, or even if the copy is synchronous or asynchronous: we just call starpu_interface_copy() according to the interface, passing it the pointers, and checking whether it returned `-EAGAIN`, which means the copy is asynchronous, and StarPU will appropriately wait for it thanks to the pointer `async_data`. This copy method is also available for 2D matrices starpu_interface_copy2d(), 3D matrices starpu_interface_copy3d(), 4D matrices starpu_interface_copy4d() and N-dim matrices starpu_interface_copynd().

starpu_interface_copy() will also manage copies between other devices such as CUDA devices, OpenCL devices, etc. But if necessary, we may manage these copies by ourselves as well. StarPU provides three functions starpu_cuda_copy_async_sync(), starpu_cuda_copy2d_async_sync() and starpu_cuda_copy3d_async_sync() that enable copying of 1D, 2D or 3D data between main memory and CUDA device memories. They first try to copy the data asynchronous, if fail or `stream` is `NULL` then copy the data synchronously. StarPU also provides several functions that are used to transfer data between RAM and OpenCL devices. starpu_opencl_copy_ram_to_opencl() copies data from RAM to an OpenCL device. starpu_opencl_copy_opencl_to_ram() copies data from an OpenCL device to RAM. starpu_opencl_copy_opencl_to_opencl() copies data between two OpenCL devices. starpu_opencl_copy_async_sync() copies data between two devices. If `event` is `NULL`, the copy is synchronous, and checking whether `ret` is set to `-EAGAIN`, which means the copy is asynchronous.

This copy method is referenced in a structure starpu_data_copy_methods

```
static const struct starpu_data_copy_methods complex_copy_methods =
{
    .any_to_any = copy_any_to_any
};
```

which was referenced in the structure starpu_data_interface_ops above.

Other fields of starpu_data_copy_methods allow providing optimized variants, notably for the case of 2D or 3D matrix tiles with non-trivial ld.

We can call starpu_interface_data_copy() to record in offline execution traces the copy.

When an asynchronous implementation of the data transfer is implemented, we can call starpu_interface_start_driver_copy_async() and starpu_interface_end_driver_copy_async() to initiate and complete asynchronous data transfers between main memory and GPU memory.

34.9.5 Data pack/peek/unpack

The copy methods allow for RAM/GPU transfers, but is not enough for e.g. transferring over MPI. That requires defining the pack/peek/unpack methods. The principle is that the method starpu_data_interface_ops::pack_data concatenates the buffer data into a newly-allocated contiguous bytes array, conversely starpu_data_interface_ops::peek_data extracts from a bytes array into the buffer data, and starpu_data_interface_ops::unpack_data does the same as starpu_data_interface_ops::peek_data but also frees the bytes array.

```
static int complex_pack_data(starpu_data_handle_t handle, unsigned node, void **ptr, starpu_ssize_t *count)
{
    STARPU_ASSERT(starpu_data_test_if_allocated_on_node(handle, node));
    struct starpu_complex_interface *complex_interface = (struct starpu_complex_interface *)
        starpu_data_get_interface_on_node(handle, node);
    *count = complex_get_size(handle);
    if (ptr != NULL)
    {
        char *data;
        data = (void*) starpu_malloc_on_node_flags(node, *count, 0);
        *ptr = data;
        memcpy(data, complex_interface->real, complex_interface->nx*sizeof(double));
        memcpy(data+complex_interface->nx*sizeof(double), complex_interface->imaginary,
      complex_interface->nx*sizeof(double));
    }
    return 0;
}
```

`complex_pack_data()` first computes the size to be allocated, then allocates it, and copies over into it the content of the two real and imaginary arrays.

```
static int complex_peek_data(starpu_data_handle_t handle, unsigned node, void *ptr, size_t count)
{
    char *data = ptr;
    STARPU_ASSERT(starpu_data_test_if_allocated_on_node(handle, node));
    struct starpu_complex_interface *complex_interface = (struct starpu_complex_interface *)
        starpu_data_get_interface_on_node(handle, node);
    STARPU_ASSERT(count == 2 * complex_interface->nx * sizeof(double));
    memcpy(complex_interface->real, data, complex_interface->nx*sizeof(double));
    memcpy(complex_interface->imaginary, data+complex_interface->nx*sizeof(double),
      complex_interface->nx*sizeof(double));
    return 0;
}
```

`complex_peek_data()` simply uses `memcpy()` to copy over from the bytes array into the data buffer.

```
static int complex_unpack_data(starpu_data_handle_t handle, unsigned node, void *ptr, size_t count)
{
    complex_peek_data(handle, node, ptr, count);
    starpu_free_on_node_flags(node, (uintptr_t) ptr, count, 0);
    return 0;
}
```

And `complex_unpack_data()` just calls `complex_peek_data()` and releases the bytes array.

34.9.6 Pointers inside the data interface

In the example described above, the two pointers stored in the data interface are data buffers, which may point into main memory, GPU memory, etc. One may also want to store pointers to meta-data for the interface, for instance the list of dimensions sizes for the n-dimension matrix interface, but such pointers are to be handled completely differently. More examples are provided in `src/datawizard/interfaces/*_interface.c`
More precisely, there are two types of pointers:

- Data pointers, which point to the actual data in RAM/GPU/etc. memory. They may be NULL when the data is not allocated (yet). StarPU will automatically call starpu_data_interface_ops::allocate_data_on_node to allocate the data pointers whenever needed, and call starpu_data_interface_ops::free_data_on_node when memory gets scarce. For instance, for the n-dimension matrix interface the pointers to the actual data (`ptr`, `dev_handle`, `offset`) are data pointers.

- Meta-data pointers, which always point to RAM memory. They are usually always allocated so that they can always be used. For instance, for the n-dimension matrix interface the array of dimension sizes and the array of ld are meta-data pointers.

This means that:

- The starpu_data_interface_ops::register_data_handle method has to allocate the meta-data pointers. If users provided a buffer for the initial value of the handle, starpu_data_interface_ops::register_data_handle sets the data pointers of the home_node interface to that buffer.
- The interface can additionally provide a `ptr_register` helper to set the data pointer of a given node. One can call starpu_data_ptr_register() to realise.
- The starpu_data_interface_ops::unregister_data_handle method has to deallocate the meta-data pointers
- The starpu_data_interface_ops::allocate_data_on_node method has to allocate the data pointers on the given node.
- The starpu_data_interface_ops::free_data_on_node method has to deallocate the data pointers on the given node.
- The starpu_data_interface_ops::cache_data_on_node transfers the data pointers from a source interface to a cached interface. This can notably take the opportunity to clear pointers in the source interface. This also needs to copy the properties that starpu_data_interface_ops::compare (or starpu_data_interface_ops::alloc_compare if defined) needs for comparing interfaces for caching compatibility.
- The starpu_data_interface_ops::reuse_data_on_node transfers the data pointers from a cached interface to the destination interface.
- The starpu_data_interface_ops::map_data has to map the data pointers on the given node. One should define function starpu_interface_map() to set this field.
- The starpu_data_interface_ops::unmap_data has to unmap the data pointers on the given node. One should define function starpu_interface_unmap() to set this field.
- The starpu_data_interface_ops::update_map has to update the data pointers on the given node. One should define function starpu_interface_update_map() to set this field.
- The filtering functions have to allocate the meta-data pointers for the child interface, and when the parent interface has data pointers, it has to set the child data pointers to point into the parent data buffers.

Note: for compressed matrices such as CSR, BCSR, COO, the `colind` and `rowptr` arrays are not meta-data pointers, but data pointers like `nzval`, because they need to be available in GPU memory for the GPU kernels.
Note: when the interface does not contain meta-data pointers, starpu_data_interface_ops::reuse_data_on_node does not need to be implemented, StarPU will just use a memcpy. Otherwise, either starpu_data_interface_ops::reuse_data_on_node must be used to transfer only the data pointers and not the meta-data pointers, or the allocation cache should be disabled by setting starpu_data_interface_ops::dontcache to 1.
Note: It should be noted that because of the allocation cache, starpu_data_interface_ops::free_data_on_node may be called on an interface which is not attached to a handle anymore. This means that the meta-data pointers will have been deallocated by starpu_data_interface_ops::unregister_data_handle, and cannot be used by starpu_data_interface_ops::free_data_on_node to e.g. compute the size to be deallocated. For instance, the n-dimension matrix interface uses an additional scalar allocsize field to store the allocation size, thus still available even when the interface is in the allocation cache.
Note: if starpu_data_interface_ops::unregister_data_handle is implemented and checks that pointers are NULL, starpu_data_interface_ops::cache_data_on_node needs to be implemented to clear the pointers when caching the allocation.

34.9.7 Helpers

We can get the unique identifier of the interface associated with the data handle by calling starpu_data_get_interface_id(), and get the next available identifier for a newly created data interface by calling starpu_data_interface_get_next_id().

34.10 The Multiformat Interface

It may be interesting to represent the same piece of data using two different data structures: one only used on CPUs, and one only used on GPUs. This can be done by using the multiformat interface. StarPU will be able to convert data from one data structure to the other when needed. Note that the scheduler `dmda` is the only one optimized for this interface. Users must provide StarPU with conversion codelets:

```
#define NX 1024
struct point array_of_structs[NX];
starpu_data_handle_t handle;
/*
 * The conversion of a piece of data is itself a task, though it is created,
 * submitted and destroyed by StarPU internals and not by the user.  Therefore,
 * we have to define two codelets.
 * Note that for now the conversion from the CPU format to the GPU format has to
 * be executed on the GPU, and the conversion from the GPU to the CPU has to be
 * executed on the CPU.
 */
#ifdef STARPU_USE_OPENCL
void cpu_to_opencl_opencl_func(void *buffers[], void *args);
struct starpu_codelet cpu_to_opencl_cl =
{
    .where = STARPU_OPENCL,
    .opencl_funcs = { cpu_to_opencl_opencl_func },
    .nbuffers = 1,
    .modes = { STARPU_RW }
};
void opencl_to_cpu_func(void *buffers[], void *args);
struct starpu_codelet opencl_to_cpu_cl =
{
    .where = STARPU_CPU,
    .cpu_funcs = { opencl_to_cpu_func },
    .cpu_funcs_name = { "opencl_to_cpu_func" },
    .nbuffers = 1,
    .modes = { STARPU_RW }
};
#endif
struct starpu_multiformat_data_interface_ops format_ops =
{
#ifdef STARPU_USE_OPENCL
    .opencl_elemsize = 2 * sizeof(float),
    .cpu_to_opencl_cl = &cpu_to_opencl_cl,
    .opencl_to_cpu_cl = &opencl_to_cpu_cl,
#endif
    .cpu_elemsize = 2 * sizeof(float),
    ...
};
starpu_multiformat_data_register(handle, STARPU_MAIN_RAM, &array_of_structs, NX, &format_ops);
```

Kernels can be written almost as for any other interface. Note that STARPU_MULTIFORMAT_GET_CPU_PTR shall only be used for CPU kernels. CUDA kernels must use STARPU_MULTIFORMAT_GET_CUDA_PTR, and OpenCL kernels must use STARPU_MULTIFORMAT_GET_OPENCL_PTR. STARPU_MULTIFORMAT_GET_NX may be used in any kind of kernel.

```
static void
multiformat_scal_cpu_func(void *buffers[], void *args)
{
    struct point *aos;
    unsigned int n;
    aos = STARPU_MULTIFORMAT_GET_CPU_PTR(buffers[0]);
    n = STARPU_MULTIFORMAT_GET_NX(buffers[0]);
    ...
}
extern "C" void multiformat_scal_cuda_func(void *buffers[], void *_args)
{
    unsigned int n;
    struct struct_of_arrays *soa;
    soa = (struct struct_of_arrays *) STARPU_MULTIFORMAT_GET_CUDA_PTR(buffers[0]);
    n = STARPU_MULTIFORMAT_GET_NX(buffers[0]);
    ...
}
```

A full example may be found in `examples/basic_examples/multiformat.c`.

34.11 Specifying A Target Node For Task Data

When executing a task on GPU, for instance, StarPU would normally copy all the needed data for the tasks to the embedded memory of the GPU. It may however happen that the task kernel would rather have some of the data kept in the main memory instead of copied in the GPU, a pivoting vector for instance. This can be achieved by setting the flag starpu_codelet::specific_nodes to `1`, and then fill the array starpu_codelet::nodes (or starpu_codelet::dyn_nodes when starpu_codelet::nbuffers is greater than STARPU_NMAXBUFS) with the node numbers where data should be copied to, or STARPU_SPECIFIC_NODE_LOCAL to let StarPU copy it to the memory node where the task will be executed.

The function starpu_task_get_current_data_node() can be used to retrieve the memory node associated with the current task being executed.

STARPU_SPECIFIC_NODE_CPU can also be used to request data to be put in CPU-accessible memory (and let StarPU choose the NUMA node). STARPU_SPECIFIC_NODE_FAST and STARPU_SPECIFIC_NODE_SLOW can also be used

For instance, with the following codelet:

```
struct starpu_codelet cl =
{
    .cuda_funcs = { kernel },
    .nbuffers = 2,
    .modes = {STARPU_RW, STARPU_RW},
    .specific_nodes = 1,
    .nodes = {STARPU_SPECIFIC_NODE_CPU, STARPU_SPECIFIC_NODE_LOCAL},
};
```

the first data of the task will be kept in the CPU memory, while the second data will be copied to the CUDA GPU as usual. A working example is available in `tests/datawizard/specific_node.c`

With the following codelet:

```
struct starpu_codelet cl =
{
    .cuda_funcs = { kernel },
    .nbuffers = 2,
    .modes = {STARPU_RW, STARPU_RW},
    .specific_nodes = 1,
    .nodes = {STARPU_SPECIFIC_NODE_LOCAL, STARPU_SPECIFIC_NODE_SLOW},
};
```

The first data will be copied into fast (but probably size-limited) local memory, while the second data will be left in slow (but large) memory. This makes sense when the kernel does not make so many accesses to the second data, and thus data being remote e.g. over a PCI bus is not a performance problem, and avoids filling the fast local memory with data which does not need the performance.

In cases where the kernel is fine with some data being either local or in the main memory, STARPU_SPECIFIC_NODE_LOCAL_OR_C can be used. StarPU will then be free to leave the data in the main memory and let the kernel access it from accelerators, or to move it to the accelerator before starting the kernel, for instance:

```
struct starpu_codelet cl =
{
    .cuda_funcs = { kernel },
    .nbuffers = 2,
    .modes = {STARPU_RW, STARPU_R},
    .specific_nodes = 1,
    .nodes = {STARPU_SPECIFIC_NODE_LOCAL, STARPU_SPECIFIC_NODE_LOCAL_OR_CPU},
};
```

An example for specifying target node is available in `tests/datawizard/specific_node.c`.

Chapter 35

Advanced Scheduling

35.1 Energy-based Scheduling

Note: by default, StarPU does not let CPU workers sleep, to let them react to task release as quickly as possible. For idle time to really let CPU cores save energy, one needs to use the `configure` option --enable-blocking-drivers.
If the application can provide some energy consumption performance model (through the field starpu_codelet::energy_model), StarPU will take it into account when distributing tasks. The target function that the scheduler **dmda** minimizes becomes `alpha * T_execution + beta * T_data_transfer + gamma * Consumption` , where `Consumption` is the estimated task consumption in Joules. To tune this parameter, use `export STARPU_SCHED_GAMMA=3000` (STARPU_SCHED_GAMMA) for instance, to express that each Joule (i.e. kW during 1000us) is worth 3000us execution time penalty. Setting `alpha` and `beta` to zero permits to only take into account energy consumption.
This is however not sufficient to correctly optimize energy: the scheduler would simply tend to run all computations on the most energy-conservative processing unit. To account for the consumption of the whole machine (including idle processing units), the idle power of the machine should be given by setting `export STARPU_IDLE_POWER=200` (STARPU_IDLE_POWER) for 200W, for instance. This value can often be obtained from the machine power supplier, e.g. by running

```
ipmitool -I lanplus -H mymachine-ipmi -U myuser -P mypasswd sdr type Current
```

The energy actually consumed by the total execution can be displayed by setting `export STARPU_PROFILING=1 STARPU_WORKER_STATS=1` (STARPU_PROFILING and STARPU_WORKER_STATS).
For OpenCL devices, on-line task consumption measurement is currently supported through the OpenCL extension `CL_PROFILING_POWER_CONSUMED`, implemented in the MoviSim simulator.
For CUDA devices, on-line task consumption measurement is supported on V100 cards and beyond. This however only works for quite long tasks, since the measurement granularity is about 10ms.
Applications can however provide explicit measurements by feeding the energy performance model by hand. Fine-grain measurement is often not feasible with the feedback provided by the hardware, so users can for instance run a given task a thousand times, measure the global consumption for that series of tasks, divide it by a thousand, repeat for varying kinds of tasks and task sizes, and eventually feed StarPU with these manual measurements. For CUDA devices starting with V100, the starpu_energy_start() and starpu_energy_stop() helpers, described in Measuring energy and power with StarPU below, make it easy.
For older models, one can use `nvidia-smi -q -d POWER` to get the current consumption in Watt. Multiplying this value by the average duration of a single task gives the consumption of the task in Joules, which can be given to starpu_perfmodel_update_history(). (examplified in Performance Model Example with the performance model `energy_model`).
Another way to provide the energy performance is to define a perfmodel with starpu_perfmodel::type STARPU_PER_ARCH or STARPU_PER_WORKER , and set the field starpu_perfmodel::arch_cost_function or starpu_perfmodel::worker_cost_function to a function which shall return the estimated consumption of the task in Joules. Such a function can for instance use starpu_task_expected_length() on the task (in µs), multiplied by the typical power consumption of the device, e.g. in W, and divided by 1000000. to get Joules. An example is in the file `tests/perfmodels/regression_based_energy.c`.
There are other functions in StarPU that are used to measure the energy consumed by the system during execution. The starpu_energy_use() function declares that there are the energy consumptions of the task, while the starpu_energy_used() function returns the total energy consumed since the start of measurement.

35.1.1 Measuring energy and power with StarPU

We have extended the performance model of StarPU to measure energy and power values of CPUs. These values are measured using the existing Performance API (PAPI) analysis library. PAPI provides the tool designer and application engineer with a consistent interface and methodology for use of the performance counter hardware found in most major microprocessors. PAPI enables software engineers to see, in near real time, the relation between software performance and processor events.

- To measure energy consumption of CPUs, we use the `RAPL` events, which are available on CPU architecture: `RAPL_ENERGY_PKG` that represents the whole CPU socket power consumption, and `RAPL_ENERGY_DRAM` that represents the RAM power consumption.

PAPI provides a generic, portable interface for the hardware performance counters available on all modern CPUs and some other components of interest that are scattered across the chip and system.
In order to use the right `rapl events` for energy measurement, user should check the `rapl events` available on the machine, using this command:

```
$ papi_native_avail
```

Depending on the system configuration, users may have to run this as **root** to get the performance counter values.
Since the measurement is for all the CPUs and the memory, the approach taken here is to run a series of tasks on all of them and to take the overall measurement.

- The example below illustrates the energy and power measurements, using the functions starpu_energy_start() and starpu_energy_stop().

In this example, we launch several tasks of the same type in parallel. To perform the energy requirement measurement of a program, we call starpu_energy_start(), which initializes energy measurement counters and starpu_energy_stop(struct starpu_perfmodel *model, struct starpu_task *task, unsigned nimpl, unsigned ntasks, int workerid, enum to stop counting and update the performance model. This ends up yielding the average energy requirement of a single task. The example below illustrates this for a given task type.

```
  unsigned N = starpu_cpu_worker_get_count() * 40;
  starpu_energy_start(-1, STARPU_CPU_WORKER);
  for (i = 0; i < N; i++)
starpu_task_insert(&cl, STARPU_EXECUTE_WHERE, STARPU_CPU, STARPU_R, arg1, STARPU_RW, arg2, 0);
  starpu_task_t *specimen = starpu_task_build(&cl, STARPU_R, arg1, STARPU_RW, arg2, 0);
  starpu_energy_stop(&codelet.energy_model, specimen, 0, N, -1, STARPU_CPU_WORKER);
 .  .  .
```

The example starts 40 times more tasks of the same type than there are CPU execution units. Once the tasks are distributed over all CPUs, the latter are all executing the same type of tasks (with the same data size and parameters); each CPU will in the end execute 40 tasks. A specimen task is then constructed and passed to starpu_energy_stop(), which will fold into the performance model the energy requirement measurement for that type and size of task.
For the energy and power measurements, depending on the system configuration, users may have to run applications as **root** to use PAPI library.
The function starpu_energy_stop() uses `PAPI_stop()` to stop counting and store the values into the array. We calculate both energy in `Joules` and power consumption in `Watt`. We call the function starpu_perfmodel_update_history() in the performance model to provide explicit measurements.

- In the CUDA case, nvml provides per-GPU energy measurement. We can thus calibrate the performance models per GPU:

```
    unsigned N = 40;
for (i = 0; i < starpu_cuda_worker_get_count(); i++) {
    int workerid = starpu_worker_get_by_type(STARPU_CUDA_WORKER, i);
    starpu_energy_start(workerid, STARPU_CUDA_WORKER);
    for (i = 0; i < N; i++)
      starpu_task_insert(&cl, STARPU_EXECUTE_ON_WORKER, workerid, STARPU_R, arg1, STARPU_RW, arg2, 0);
    starpu_task_t *specimen = starpu_task_build(&cl, STARPU_R, arg1, STARPU_RW, arg2, 0);
    starpu_energy_stop(&codelet.energy_model, specimen, 0, N, workerid, STARPU_CUDA_WORKER);
   }
```

- A complete example is available in `tests/perfmodels/regression_based_memset.c`

35.2 Static Scheduling

In some cases, one may want to force some scheduling, for instance force a given set of tasks to GPU0, another set to GPU1, etc. while letting some other tasks be scheduled on any other device. This can indeed be useful to guide StarPU into some work distribution, while still letting some degree of dynamism. For instance, to force execution of a task on CUDA0:

```
task->execute_on_a_specific_worker = 1;
task->workerid = starpu_worker_get_by_type(STARPU_CUDA_WORKER, 0);
```

An example is in the file `tests/errorcheck/invalid_tasks.c`.

or equivalently

```
starpu_task_insert(&cl, ..., STARPU_EXECUTE_ON_WORKER, starpu_worker_get_by_type(STARPU_CUDA_WORKER, 0),
      ...);
```

One can also specify a set of worker(s) which are allowed to take the task, as an array of bit, for instance to allow workers 2 and 42:

```
task->workerids = calloc(2,sizeof(uint32_t));
task->workerids[2/32] |= (1 « (2%32));
task->workerids[42/32] |= (1 « (42%32));
task->workerids_len = 2;
```

One can also specify the order in which tasks must be executed by setting the field starpu_task::workerorder. An example is available in the file `tests/main/execute_schedule.c`. If this field is set to a non-zero value, it provides the per-worker consecutive order in which tasks will be executed, starting from 1. For a given of such task, the worker will thus not execute it before all the tasks with smaller order value have been executed, notably in case those tasks are not available yet due to some dependencies. This eventually gives total control of task scheduling, and StarPU will only serve as a "self-timed" task runtime. Of course, the provided order has to be runnable, i.e. a task should not depend on another task bound to the same worker with a bigger order.

Note however that using scheduling contexts while statically scheduling tasks on workers could be tricky. Be careful to schedule the tasks exactly on the workers of the corresponding contexts, otherwise the workers' corresponding scheduling structures may not be allocated or the execution of the application may deadlock. Moreover, the hypervisor should not be used when statically scheduling tasks.

35.3 Configuring Heteroprio

Within Heteroprio, one priority per processing unit type is assigned to each task, such that a task has several priorities. Each worker pops the task that has the highest priority for the hardware type it uses, which could be CPU or CUDA for example. Therefore, the priorities has to be used to manage the critical path, but also to promote the consumption of tasks by the more appropriate workers.

The tasks are stored inside buckets, where each bucket corresponds to a priority set. Then each worker uses an indirect access array to know the order in which it should access the buckets. Moreover, all the tasks inside a bucket must be compatible with all the processing units that may access it (at least).

These priorities are now automatically assigned by Heteroprio in auto calibration mode using heuristics. If you want to set these priorities manually, you can change STARPU_HETEROPRIO_USE_AUTO_CALIBRATION and follow the example below.

In this example code, we have 5 types of tasks. CPU workers can compute all of them, but CUDA workers can only execute tasks of types 0 and 1, and are expected to go 20 and 30 time faster than the CPU, respectively.

```
#include <starpu_heteroprio.h>
 // Before calling starpu_init
struct starpu_conf conf;
starpu_conf_init(&conf);
 // Inform StarPU to use Heteroprio
conf.sched_policy_name = "heteroprio";
 // Inform StarPU about the function that will init the priorities in Heteroprio
 // where init_heteroprio is a function to implement
conf.sched_policy_callback = &init_heteroprio;
 // Do other things with conf if needed, then init StarPU
starpu_init(&conf);
void init_heteroprio(unsigned sched_ctx) {
  // CPU uses 5 buckets and visits them in the natural order
  starpu_heteroprio_set_nb_prios(sched_ctx, STARPU_CPU_WORKER, 5);
  // It uses direct mapping idx => idx
  for(unsigned idx = 0; idx < 5; ++idx){
    starpu_heteroprio_set_mapping(sched_ctx, STARPU_CPU_WORKER, idx, idx);
    // If there is no CUDA worker we must tell that CPU is faster
    starpu_heteroprio_set_faster_arch(sched_ctx, STARPU_CPU_WORKER, idx);
  }
  if(starpu_cuda_worker_get_count()){
    // CUDA is enabled and uses 2 buckets
    starpu_heteroprio_set_nb_prios(sched_ctx, STARPU_CUDA_WORKER, 2);
    // CUDA will first look at bucket 1
    starpu_heteroprio_set_mapping(sched_ctx, STARPU_CUDA_WORKER, 0, 1);
    // CUDA will then look at bucket 2
    starpu_heteroprio_set_mapping(sched_ctx, STARPU_CUDA_WORKER, 1, 2);
    // For bucket 1 CUDA is the fastest
    starpu_heteroprio_set_faster_arch(sched_ctx, STARPU_CUDA_WORKER, 1);
    // And CPU is 30 times slower
    starpu_heteroprio_set_arch_slow_factor(sched_ctx, STARPU_CPU_WORKER, 1, 30.0f);
    // For bucket 0 CUDA is the fastest
    starpu_heteroprio_set_faster_arch(sched_ctx, STARPU_CUDA_WORKER, 0);
    // And CPU is 20 times slower
    starpu_heteroprio_set_arch_slow_factor(sched_ctx, STARPU_CPU_WORKER, 0, 20.0f);
  }
}
```

Then, when a task is inserted, **the priority of the task will be used to select in which bucket is has to be stored**. So, in the given example, the priority of a task will be between 0 and 4 included. However, tasks of priorities 0-1 must provide CPU and CUDA kernels, and tasks of priorities 2-4 must provide CPU kernels (at least). The full source code of this example is available in the file `examples/scheduler/heteroprio_test.c`

35.3.1 Using locality aware Heteroprio

Heteroprio supports a mode where locality is evaluated to guide the distribution of the tasks (see https://peerj.com/articles/cs-190.pdf). Currently, this mode is available using the dedicated function or an environment variable STARPU_HETEROPRIO_USE_LA, and can be configured using environment variables.

```
void starpu_heteroprio_set_use_locality(unsigned sched_ctx_id, unsigned use_locality);
```

In this mode, multiple strategies are available to determine which memory node's workers are the most qualified for executing a specific task. This strategy can be set with STARPU_LAHETEROPRIO_PUSH and available strategies are:

- WORKER: the worker which pushed the task is preferred for the execution.
- LcS: the node with the shortest data transfer time (estimated by StarPU) is the most qualified
- LS_SDH: the node with the smallest data amount to be transferred will be preferred.
- LS_SDH2: similar to LS_SDH, but data in write access is counted in a quadratic manner to give them more importance.
- LS_SDHB: similar to LS_SDH, but data in write access is balanced with a coefficient (its value is set to 1000) and for the same amount of data, the one with fewer pieces of data to be transferred will be preferred.
- LC_SMWB: similar to LS_SDH, but the amount of data in write access gets multiplied by a coefficient which gets closer to 2 as the amount of data in read access gets larger than the data in write access.
- AUTO: strategy by default, this one selects the best strategy and changes it in runtime to improve performance

Other environment variables to configure LaHeteteroprio are documented in Configuring LAHeteroprio

35.3.2 Using Heteroprio in auto-calibration mode

In this mode, Heteroprio saves data about each program execution, in order to improve future ones. By default, these files are stored in the folder used by perfmodel, but this can be changed using the STARPU_HETEROPRIO_DATA_DIR environment variable. You can also specify the data filename directly using STARPU_HETEROPRIO_DATA_FILE.

Additionally, to assign priorities to tasks, Heteroprio needs a way to detect that some tasks are similar. By default, Heteroprio looks for tasks with the same perfmodel, or with the same codelet's name if no perfmodel was assigned. This behavior can be changed to only consider the codelet's name by setting STARPU_HETEROPRIO_CODELET_GROUPING_STRATEGY to `1`

Other environment variables to configure AutoHeteteroprio are documented in Configuring AutoHeteroprio

Chapter 36

Scheduling Contexts

TODO: improve!

36.1 General Ideas

Scheduling contexts represent abstracts sets of workers that allow the programmers to control the distribution of computational resources (i.e. CPUs and GPUs) to concurrent kernels. The main goal is to minimize interferences between the execution of multiple parallel kernels, by partitioning the underlying pool of workers using contexts. Scheduling contexts additionally allow a user to make use of a different scheduling policy depending on the target resource set.

36.2 Creating A Context

By default, the application submits tasks to an initial context, which disposes of all the computation resources available to StarPU (all the workers). If the application programmer plans to launch several kernels simultaneously, by default these kernels will be executed within this initial context, using a single scheduler policy (see Task Scheduling Policies). Meanwhile, if the application programmer is aware of the demands of these kernels and of the specificity of the machine used to execute them, the workers can be divided between several contexts. These scheduling contexts will isolate the execution of each kernel, and they will permit the use of a scheduling policy proper to each one of them.

Scheduling Contexts may be created in two ways: either the programmers indicates the set of workers corresponding to each context (providing he knows the identifiers of the workers running within StarPU), or the programmer does not provide any worker list and leaves the Hypervisor to assign workers to each context according to their needs (Scheduling Context Hypervisor).

Both cases require a call to the function starpu_sched_ctx_create(), which requires as input the worker list (the exact list or a `NULL` pointer), the amount of workers (or `-1` to designate all workers on the platform) and a list of optional parameters such as the scheduling policy, terminated by a `0`. The scheduling policy can be a character list corresponding to the name of a StarPU predefined policy or the pointer to a custom policy. The function returns an identifier of the context created, which you will use to indicate the context you want to submit the tasks to. A basic example is available in the file `examples/sched_ctx/sched_ctx.c`.

```
/* the list of resources the context will manage */
int workerids[3] = {1, 3, 10};
/* indicate the list of workers assigned to it, the number of workers,
the name of the context and the scheduling policy to be used within
the context */
int id_ctx = starpu_sched_ctx_create(workerids, 3, "my_ctx", STARPU_SCHED_CTX_POLICY_NAME, "dmda", 0);
/* let StarPU know that the following tasks will be submitted to this context */
starpu_sched_ctx_set_context(id);
/* submit the task to StarPU */
starpu_task_submit(task);
```

Note: Parallel greedy and parallel heft scheduling policies do not support the existence of several disjoint contexts on the machine. Combined workers are constructed depending on the entire topology of the machine, not only the one belonging to a context.

36.2.1 Creating A Context With The Default Behavior

If **no scheduling policy** is specified when creating the context, it will be used as **another type of resource**: a parallel worker. A parallel worker is a context without scheduler (eventually delegated to another runtime). For more information, see Creating Parallel Workers On A Machine. It is therefore **mandatory** to stipulate a scheduler to use the contexts in this traditional way.

To create a **context** with the default scheduler, that is either controlled through the environment variable `STARPU_SCHED` or the StarPU default scheduler, one can explicitly use the option `STARPU_SCHED_CTX_POLICY_NAME, ""` as in the following example:

```
/* the list of resources the context will manage */
int workerids[3] = {1, 3, 10};
/* indicate the list of workers assigned to it, the number of workers,
and use the default scheduling policy.  */
int id_ctx = starpu_sched_ctx_create(workerids, 3, "my_ctx", STARPU_SCHED_CTX_POLICY_NAME, "", 0);
/* ....  */
```

A full example is available in the file `examples/sched_ctx/two_cpu_contexts.c`.

36.3 Creating A Context To Partition a GPU

The contexts can also be used to group a set of SMs of an NVIDIA GPU in order to isolate the parallel kernels and allow them to coexecution on a specified partition of the GPU.

Each context will be mapped to a stream and users can indicate the number of SMs. The context can be added to a larger context already grouping CPU cores. This larger context can use a scheduling policy that assigns tasks to both CPUs and contexts (partitions of the GPU) based on performance models adjusted to the number of SMs.

The GPU implementation of the task has to be modified accordingly and receive as a parameter the number of SMs.

```
/* get the available streams (suppose we have nstreams = 2 by specifying them with STARPU_NWORKER_PER_CUDA=2
      */
int nstreams = starpu_worker_get_stream_workerids(gpu_devid, stream_workerids, STARPU_CUDA_WORKER);
int sched_ctx[nstreams];
sched_ctx[0] = starpu_sched_ctx_create(&stream_workerids[0], 1, "subctx",  STARPU_SCHED_CTX_CUDA_NSMS, 6,
      0);
sched_ctx[1] = starpu_sched_ctx_create(&stream_workerids[1], 1, "subctx",  STARPU_SCHED_CTX_CUDA_NSMS, 7,
      0);
int ncpus = 4;
int workers[ncpus+nstreams];
workers[ncpus+0] = stream_workerids[0];
workers[ncpus+1] = stream_workerids[1];
big_sched_ctx = starpu_sched_ctx_create(workers, ncpus+nstreams, "ctx1", STARPU_SCHED_CTX_SUB_CTXS,
      sched_ctxs, nstreams, STARPU_SCHED_CTX_POLICY_NAME, "dmdas", 0);
starpu_task_submit_to_ctx(task, big_sched_ctx);
```

A full example is available in the file `examples/sched_ctx/gpu_partition.c`.

36.4 Modifying A Context

A scheduling context can be modified dynamically. The application may change its requirements during the execution, and the programmer can add additional workers to a context or remove those no longer needed. In the following example, we have two scheduling contexts `sched_ctx1` and `sched_ctx2`. After executing a part of the tasks, some of the workers of `sched_ctx1` will be moved to context `sched_ctx2`.

```
/* the list of resources that context 1 will give away */
int workerids[3] = {1, 3, 10};
/* add the workers to context 1 */
starpu_sched_ctx_add_workers(workerids, 3, sched_ctx2);
/* remove the workers from context 2 */
starpu_sched_ctx_remove_workers(workerids, 3, sched_ctx1);
```

An example is available in the file `examples/sched_ctx/sched_ctx_remove.c`.

36.5 Submitting Tasks To A Context

The application may submit tasks to several contexts, either simultaneously or sequentially. If several threads of submission are used, the function starpu_sched_ctx_set_context() may be called just before starpu_task_submit(). Thus, StarPU considers that the current thread will submit tasks to the corresponding context. An example is available in the file `examples/sched_ctx/gpu_partition.c`.

When the application may not assign a thread of submission to each context, the id of the context must be indicated by using the function starpu_task_submit_to_ctx() or the field STARPU_SCHED_CTX for starpu_task_insert(). An example is available in the file `examples/sched_ctx/sched_ctx.c`.

36.6 Deleting A Context

When a context is no longer needed, it must be deleted. The application can indicate which context should keep the resources of a deleted one. All the tasks of the context should be executed before doing this. Thus, the programmer may use either a barrier and then delete the context directly, or just indicate that other tasks will not be submitted later on to the context (such that when the last task is executed its workers will be moved to the inheritor) and delete the context at the end of the execution (when a barrier will be used eventually).

```
/* when the context 2 is deleted context 1 inherits its resources */
starpu_sched_ctx_set_inheritor(sched_ctx2, sched_ctx1);
/* submit tasks to context 2 */
for (i = 0; i < ntasks; i++)
    starpu_task_submit_to_ctx(task[i],sched_ctx2);
/* indicate that context 2 finished submitting and that */
/* as soon as the last task of context 2 finished executing */
/* its workers can be moved to the inheritor context */
starpu_sched_ctx_finished_submit(sched_ctx1);
/* wait for the tasks of both contexts to finish */
starpu_task_wait_for_all();
/* delete context 2 */
starpu_sched_ctx_delete(sched_ctx2);
/* delete context 1 */
starpu_sched_ctx_delete(sched_ctx1);
```

A full example is available in the file `examples/sched_ctx/sched_ctx.c`.

36.7 Emptying A Context

A context may have no resources at the beginning or at a certain moment of the execution. Tasks can still be submitted to these contexts, they will be executed as soon as the contexts will have resources. A list of tasks pending to be executed is kept and will be submitted when workers are added to the contexts.

```
/* create a empty context */
unsigned sched_ctx_id = starpu_sched_ctx_create(NULL, 0, "ctx", 0);
/* submit a task to this context */
starpu_sched_ctx_set_context(&sched_ctx_id);
ret = starpu_task_insert(&codelet, 0);
STARPU_CHECK_RETURN_VALUE(ret, "starpu_task_insert");
/* add CPU workers to the context */
int procs[STARPU_NMAXWORKERS];
int nprocs = starpu_cpu_worker_get_count();
starpu_worker_get_ids_by_type(STARPU_CPU_WORKER, procs, nprocs);
starpu_sched_ctx_add_workers(procs, nprocs, sched_ctx_id);
/* and wait for the task termination */
starpu_task_wait_for_all();
```

The full example is available in the file `examples/sched_ctx/sched_ctx_empty.c`.

However, if resources are never allocated to the context, the application will not terminate. If these tasks have low priority, the application can inform StarPU to not submit them by calling the function starpu_sched_ctx_stop_task_submission().

Chapter 37

Scheduling Context Hypervisor

37.1 What Is The Hypervisor

StarPU proposes a platform to construct Scheduling Contexts, to delete and modify them dynamically. A parallel kernel, can thus be isolated into a scheduling context and interferences between several parallel kernels are avoided. If users know exactly how many workers each scheduling context needs, they can assign them to the contexts at their creation time or modify them during the execution of the program.
The Scheduling Context Hypervisor Plugin is available for users who do not dispose of a regular parallelism, who cannot know in advance the exact size of the context and need to resize the contexts according to the behavior of the parallel kernels.
The Hypervisor receives information from StarPU concerning the execution of the tasks, the efficiency of the resources, etc. and it decides accordingly when and how the contexts can be resized. Basic strategies of resizing scheduling contexts already exist, but a platform for implementing additional custom ones is available.
Several examples of hypervisor are provided in `sc_hypervisor/examples/*.c`

37.2 Start the Hypervisor

The Hypervisor must be initialized once at the beginning of the application. At this point, a resizing policy should be indicated. This strategy depends on the information the application is able to provide to the hypervisor, as well as on the accuracy needed for the resizing procedure. For example, the application may be able to provide an estimation of the workload of the contexts. In this situation, the hypervisor may decide what resources the contexts need. However, if no information is provided, the hypervisor evaluates the behavior of the resources and of the application and makes a guess about the future. The hypervisor resizes only the registered contexts. The basic example is available in the file `sc_hypervisor/examples/sched_ctx_utils/sched_ctx_utils.c`.

37.3 Interrogate The Runtime

The runtime provides the hypervisor with information concerning the behavior of the resources and the application. This is done by using the `performance_counters` which represent callbacks indicating when the resources are idle or not efficient, when the application submits tasks or when it becomes too slow.

37.4 Trigger the Hypervisor

The resizing is triggered either when the application requires it (sc_hypervisor_resize_ctxs()) or when the initial distribution of resources alters the performance of the application (the application is too slow or the resource are idle for too long time). An example is available in the file `sc_hypervisor/examples/hierarchical_ctxs/resize_hierarchical_ctxs.c`.
If the environment variable SC_HYPERVISOR_TRIGGER_RESIZE is set to `speed`, the monitored speed of the contexts is compared to a theoretical value computed with a linear program, and the resizing is triggered whenever the two values do not correspond. Otherwise, if the environment variable is set to `idle` the hypervisor triggers the resizing algorithm whenever the workers are idle for a period longer than the threshold indicated by the programmer. When this happens, different resizing strategy are applied that target minimizing the total execution of the

application, the instant speed or the idle time of the resources.

37.5 Resizing Strategies

The plugin proposes several strategies for resizing the scheduling context.
The **Application driven** strategy uses users's input concerning the moment when they want to resize the contexts. Thus, users tag the task that should trigger the resizing process. One can set directly the field starpu_task::hypervisor_tag or use the macro STARPU_HYPERVISOR_TAG in the function starpu_task_insert().

```
task.hypervisor_tag = 2;
```

or

```
starpu_task_insert(&codelet,
                   ...,
                   STARPU_HYPERVISOR_TAG, 2,
                   0);
```

Then users have to indicate that when a task with the specified tag is executed, the contexts should resize.

```
sc_hypervisor_resize(sched_ctx, 2);
```

Users can use the same tag to change the resizing configuration of the contexts if they consider it necessary.

```
sc_hypervisor_ctl(sched_ctx,
                   SC_HYPERVISOR_MIN_WORKERS, 6,
                   SC_HYPERVISOR_MAX_WORKERS, 12,
                   SC_HYPERVISOR_TIME_TO_APPLY, 2,
                   NULL);
```

The **Idleness** based strategy moves workers unused in a certain context to another one needing them. (see Scheduling Context Hypervisor - Regular usage)

```
int workerids[3] = {1, 3, 10};
int workerids2[9] = {0, 2, 4, 5, 6, 7, 8, 9, 11};
sc_hypervisor_ctl(sched_ctx_id,
           SC_HYPERVISOR_MAX_IDLE, workerids, 3, 10000.0,
           SC_HYPERVISOR_MAX_IDLE, workerids2, 9, 50000.0,
           NULL);
```

The **Gflops/s rate** based strategy resizes the scheduling contexts such that they all finish at the same time. The speed of each of them is computed and once one of them is significantly slower, the resizing process is triggered. In order to do these computations, users have to input the total number of instructions needed to be executed by the parallel kernels and the number of instruction to be executed by each task.
The number of flops to be executed by a context are passed as parameter when they are registered to the hypervisor,

```
sc_hypervisor_register_ctx(sched_ctx_id, flops)
```

and the one to be executed by each task are passed when the task is submitted. The corresponding field is starpu_task::flops and the corresponding macro in the function starpu_task_insert() is STARPU_FLOPS (**Caution**: but take care of passing a double, not an integer, otherwise parameter passing will be bogus). When the task is executed, the resizing process is triggered.

```
task.flops = 100;
```

or

```
starpu_task_insert(&codelet,
                   ...,
                   STARPU_FLOPS, (double) 100,
                   0);
```

The **Feft** strategy uses a linear program to predict the best distribution of resources such that the application finishes in a minimum amount of time. As for the **Gflops/s rate** strategy, the programmers have to indicate the total number of flops to be executed when registering the context. This number of flops may be updated dynamically during the execution of the application whenever this information is not very accurate from the beginning. The function sc_hypervisor_update_diff_total_flops() is called in order to add or to remove a difference to the flops left to be executed. Tasks are provided also the number of flops corresponding to each one of them. During the execution of the application, the hypervisor monitors the consumed flops and recomputes the time left and the number of resources to use. The speed of each type of resource is (re)evaluated and inserter in the linear program in order to better adapt to the needs of the application.
The **Teft** strategy uses a linear program too, that considers all the types of tasks and the number of each of them, and it tries to allocate resources such that the application finishes in a minimum amount of time. A previous calibration of StarPU would be useful in order to have good predictions of the execution time of each type of task.
The types of tasks may be determined directly by the hypervisor when they are submitted. However, there are applications that do not expose all the graph of tasks from the beginning. In this case, in order to let the hypervisor know about all the tasks, the function sc_hypervisor_set_type_of_task() will just inform the hypervisor about future tasks without submitting them right away.
The **Ispeed** strategy divides the execution of the application in several frames. For each frame, the hypervisor computes the speed of the contexts and tries making them run at the same speed. The strategy requires less contribution from users, as the hypervisor requires only the size of the frame in terms of flops.

```
int workerids[3] = {1, 3, 10};
int workerids2[9] = {0, 2, 4, 5, 6, 7, 8, 9, 11};
sc_hypervisor_ctl(sched_ctx_id,
                  SC_HYPERVISOR_ISPEED_W_SAMPLE, workerids, 3, 2000000000.0,
                  SC_HYPERVISOR_ISPEED_W_SAMPLE, workerids2, 9, 200000000000.0,
                  SC_HYPERVISOR_ISPEED_CTX_SAMPLE, 60000000000.0,
            NULL);
```

The **Throughput** strategy focuses on maximizing the throughput of the resources and resizes the contexts such that the machine is running at its maximum efficiency (maximum instant speed of the workers).

37.6 Defining A New Hypervisor Policy

While Scheduling Context Hypervisor Plugin comes with a variety of resizing policies (see Resizing Strategies), it may sometimes be desirable to implement custom policies to address specific problems. The API described below allows users to write their own resizing policy.

Here is an example of how to define a new policy

```
struct sc_hypervisor_policy dummy_policy =
{
       .handle_poped_task = dummy_handle_poped_task,
       .handle_pushed_task = dummy_handle_pushed_task,
       .handle_idle_cycle = dummy_handle_idle_cycle,
       .handle_idle_end = dummy_handle_idle_end,
       .handle_post_exec_hook = dummy_handle_post_exec_hook,
       .custom = 1,
       .name = "dummy"
};
```

Examples are provided in `sc_hypervisor/src/hypervisor_policies/*_policy.c`

Chapter 38

How To Define a New Scheduling Policy

38.1 Introduction

StarPU provides two ways of defining a scheduling policy, a basic monolithic way, and a modular way.
The basic monolithic way is directly connected with the core of StarPU, which means that the policy then has to handle all performance details, such as data prefetching, task performance model calibration, worker locking, etc. `examples/scheduler/dummy_sched.c` is a trivial example which does not handle this, and thus e.g. does not achieve any data prefetching or smart scheduling.
The modular way allows implementing just one component, and reuse existing components to cope with all these details. `examples/scheduler/dummy_modular_sched.c` is a trivial example very similar to `dummy_sched.c`, but implemented as a component, which allows assembling it with other components, and notably get data prefetching support for free, and task performance model calibration is properly performed, which allows to easily extend it into taking task duration into account, etc.

38.2 Helper functions for defining a scheduling policy (Basic or modular)

Make sure to have a look at the Scheduling Policy section, which provides a complete list of the functions available for writing advanced schedulers.
This includes getting an estimation for a task computation completion with starpu_task_expected_length(), for a speedup factor relative to CPU speed with starpu_worker_get_relative_speedup(), for the expected data transfer time in micro-seconds with starpu_task_expected_data_transfer_time(), starpu_task_expected_data_transfer_time_for(), or starpu_data_expected_transfer_time(), for the expected conversion time in micro-seconds with starpu_task_expected_conversion_t
for the required energy with starpu_task_expected_energy() or starpu_task_worker_expected_energy(), etc. Per-worker variants are also available with starpu_task_worker_expected_length(), etc. The average over workers is also available with starpu_task_expected_length_average() and starpu_task_expected_energy_average(). Other useful functions include starpu_transfer_bandwidth(), starpu_transfer_latency(), starpu_transfer_predict(), ... The successors of a task can be obtained with starpu_task_get_task_succs(). One can also directly test the presence of a data handle with starpu_data_is_on_node(). Prefetches can be triggered by calling either starpu_prefetch_task_input_for(), starpu_idle_prefetch_task_input_for(), starpu_prefetch_task_input_for_prio(), or starpu_idle_prefetch_task_input_for_prio(). And prefetching data on a specified node can use either starpu_prefetch_task_input_on_node(), starpu_prefetch_task_input_on_node_prio(), starpu_idle_prefetch_task_input_on_node(), or starpu_idle_prefetch_task_input_on_node_prio(). The `_prio` versions allow specifying a priority for the transfer (instead of taking the task priority by default). These prefetches are only processed when there are no fetch data requests (i.e. a task is waiting for it) to process. The `_idle` versions queue the transfers on the idle prefetch queue, which is only processed when there are no non-idle prefetches to process. starpu_get_prefetch_flag() is a convenient helper for checking the value of the STARPU_PREFETCH environment variable. When a scheduler does such prefetching, it should set the `prefetches` field of the `starpu_sched_policy` to 1, to prevent the core from triggering its own prefetching.
For applications that need to prefetch data or to perform other pre-execution setup before a task is executed, it is useful to call the function starpu_task_notify_ready_soon_register() which registers a callback function when a task is about to become ready for execution. starpu_worker_set_going_to_sleep_callback() and starpu_worker_set_waking_up_callback() allow to register an external resource manager callback function that will be notified about workers going to sleep or waking up, when StarPU is compiled with support for blocking drivers

and worker callbacks.
Schedulers should call starpu_task_set_implementation() or starpu_task_get_implementation() to specify or to retrieve the codelet implementation to be executed when executing a specific task.
One can determine if a worker type is capable of executing a specific task by calling the function starpu_worker_type_can_execute_task(). The function starpu_sched_find_all_worker_combinations() must be used to identify all viable worker combinations that can execute a parallel task. starpu_combined_worker_get_count() and starpu_worker_is_combined_worker() can be used to determine the number of different combined workers and whether a particular worker is a combined worker respectively. starpu_combined_worker_get_id() allows to get the identifier of the current combined worker. starpu_combined_worker_assign_workerid() allow users to or register a new combined worker and get its identifier, it then needs to be given to a worker collection with the starpu_worker_collection::add. starpu_combined_worker_get_desceiption() returns the description of a combined worker. Additionally, the function starpu_worker_is_blocked_in_parallel() is utilized to determine if a worker is currently blocked in a parallel task, whereas starpu_worker_is_slave_somewhere() can be called to determine if a worker is presently functioning as a slave for another worker. StarPU also provides two functions for initializing and preparing the execution of parallel tasks: starpu_parallel_task_barrier_init() and starpu_parallel_task_barrier_init_n().
Usual functions can be used on tasks, for instance one can use the following to get the data size for a task.

```
size = 0;
write = 0;
if (task->cl)
    for (i = 0; i < STARPU_TASK_GET_NBUFFERS(task); i++)
    {
        starpu_data_handle_t data = STARPU_TASK_GET_HANDLE(task, i)
        size_t datasize = starpu_data_get_size(data);
        size += datasize;
        if (STARPU_TASK_GET_MODE(task, i) & STARPU_W)
            write += datasize;
    }
```

Task queues can be implemented with the starpu_task_list functions. The function starpu_task_list_init() is used to initialize an empty list structure. Once the list is initialized, new tasks can be added to it using the starpu_task_list_push_front() and starpu_task_list_push_back() to add a task to the front or back of the list respectively. starpu_task_list_front() and starpu_task_list_back() can be used to get the first or last task in the list without removing it. starpu_task_list_begin() and starpu_task_list_end() can be used to get the task iterators from the beginning of the list and check whether it is the end of the list respectively. starpu_task_list_next() can be used to get the next task in the list, which is not erase-safe. starpu_task_list_empty() can be used to check whether the list is empty. To remove tasks from the queue, the function starpu_task_list_erase() is used to remove a specific task from the list. starpu_task_list_pop_front() and starpu_task_list_pop_back() can be used to remove the first or last task from the list. Finally, the function starpu_task_list_ismember() is used to check whether a given task is contained in the list. The function starpu_task_list_move() is used to move list from one head to another.
Access to the `hwloc` topology is available with starpu_worker_get_hwloc_obj().

38.3 Defining A New Basic Scheduling Policy

A full example showing how to define a new scheduling policy is available in the StarPU sources in `examples/scheduler/dummy_sched.c`.
The scheduler has to provide methods:

```
static struct starpu_sched_policy dummy_sched_policy =
{
    .init_sched = init_dummy_sched,
    .deinit_sched = deinit_dummy_sched,
    .add_workers = dummy_sched_add_workers,
    .remove_workers = dummy_sched_remove_workers,
    .push_task = push_task_dummy,
    .pop_task = pop_task_dummy,
    .policy_name = "dummy",
    .policy_description = "dummy scheduling strategy"
};
```

The idea is that when a task becomes ready for execution, the starpu_sched_policy::push_task method is called to give the ready task to the scheduler. Then call starpu_push_task_end() to notify that the specified task has been pushed. When a worker is idle, the starpu_sched_policy::pop_task method is called to get a task from the scheduler. It is up to the scheduler to implement what is between. A simple eager scheduler is for instance to make starpu_sched_policy::push_task push the task to a global list, and make starpu_sched_policy::pop_task pop from this list. A scheduler can also use starpu_push_local_task() to directly push tasks to a per-worker queue, and then StarPU does not even need to implement starpu_sched_policy::pop_task. If there are no ready tasks within the scheduler, it can just return `NULL`, and the worker will sleep.

starpu_sched_policy::add_workers and starpu_sched_policy::remove_workers are used to add or remove workers to or from a scheduling policy, so that the number of workers in a policy can be dynamically adjusted. After adding or removing workers from a scheduling policy, the worker task lists should be updated to ensure that the workers are assigned tasks appropriately. By calling starpu_sched_ctx_worker_shares_tasks_lists(), you can specify whether a worker may pop tasks from the task list of other workers or if there is a central list with tasks for all the workers.

The starpu_sched_policy section provides the exact rules that govern the methods of the policy.

One can enumerate the workers with this iterator:

```
struct starpu_worker_collection *workers = starpu_sched_ctx_get_worker_collection(sched_ctx_id);
struct starpu_sched_ctx_iterator it;
workers->init_iterator(workers, &it);
while(workers->has_next(workers, &it))
{
        unsigned worker = workers->get_next(workers, &it);
        ...
}
```

To provide synchronization between workers, a per-worker lock exists to protect the data structures of a given worker. It is acquired around scheduler methods, so that the scheduler does not need any additional mutex to protect its per-worker data.

In case the scheduler wants to access another scheduler's data, it should use starpu_worker_lock() and starpu_worker_unlock(), or use starpu_worker_trylock() which will not block if the lock is not immediately available, or use starpu_worker_lock_self() and starpu_worker_unlock_self() to acquire and to release a lock on the worker associated with the current thread.

Calling

```
starpu_worker_lock(B)
```

from a worker `A` will however thus make worker `A` wait for worker `B` to complete its scheduling method. That may be a problem if that method takes a long time, because it is e.g. computing a heuristic or waiting for another mutex, or even cause deadlocks if worker `B` is calling

```
starpu_worker_lock(A)
```

at the same time. In such a case, worker `B` must call starpu_worker_relax_on() and starpu_worker_relax_off() around the section which potentially blocks (and does not actually need protection). While a worker is in relaxed mode, e.g. between a pair of starpu_worker_relax_on() and starpu_worker_relax_off() calls, its state can be altered by other threads: for instance, worker `A` can push tasks for worker `B`. In consequence, worker `B` must re-assess its state after

```
starpu_worker_relax_off(B)
```

, such as taking possible new tasks pushed to its queue into account. Calling starpu_worker_get_relax_state() to query the relaxation state of a worker.

When the starpu_sched_policy::push_task method has pushed a task for another worker, one has to call starpu_wake_worker_relax(), starpu_wake_worker_relax_light(), starpu_wake_worker_no_relax() or starpu_wake_worker_locked() so that the worker wakes up and picks it. If the task was pushed on a shared queue, one may want to only wake one idle worker. An example doing this is available in `src/sched_policies/eager_central_policy.c`. When the scheduling policy makes a scheduling decision for a task, it shouhld call starpu_sched_task_break().

Schedulers can set the minimum or maximum task priority level supported by the scheduling policy by calling starpu_sched_set_min_priority() or starpu_sched_set_max_priority(), and then applications can call starpu_sched_get_min_priority() or starpu_sched_get_max_priority() to retrieve the minimum or maximum priority value. The file `src/sched_policies/heteroprio.c` shows how to uses these functions.

When scheduling a task, it is important to check whether the specified worker can execute the codelet before assigning the task to that worker. This is done using the starpu_worker_can_execute_task() function, or starpu_combined_worker_can_execute_task() which is compatible with combined workers, or starpu_worker_can_execute_task_impl() which also returns the list of implementation numbers that can be used by the worker to execute the task, or starpu_worker_can_execute_task_first_impl() which also returns the first implementation number that can be used.

A pointer to one data structure specific to the scheduler can be set with starpu_sched_ctx_set_policy_data() and fetched with starpu_sched_ctx_get_policy_data(). Per-worker data structures can then be stored in it by allocating a STARPU_NMAXWORKERS -sized array of structures indexed by workers.

A variety of examples of advanced schedulers can be read in `src/sched_policies`, for instance `random_policy.c`, `eager_central_policy.c`, `work_stealing_policy.c` Code protected by `if (_starpu_get_nsched_ctxs() > 1)` can be ignored, this is for scheduling contexts, which is an experimental feature.

38.4 Defining A New Modular Scheduling Policy

StarPU's Modularized Schedulers are made of individual Scheduling Components Modularizedly assembled as a Scheduling Tree. Each Scheduling Component has a unique purpose, such as prioritizing tasks or mapping tasks over resources. A typical Scheduling Tree is shown below.

```
                                 |
             starpu_push_task    |
                                 |
                                 v
                           Fifo_Component
                                |  ^
                        Push    |  |    Can_Push
                                v  |
                          Eager_Component
                                |  ^
                                |  |
                                v  |
              --------><-------------------><---------
              |  ^                                |  ^
      Push    |  |    Can_Push            Push    |  |    Can_Push
              v  |                                v  |
         Fifo_Component                       Fifo_Component
              |  ^                                |  ^
      Pull    |  |    Can_Pull            Pull    |  |    Can_Pull
              v  |                                v  |
        Worker_Component                     Worker_Component
                  |                             |
starpu_pop_task   |                             |
                  v                             v
```

When a task is pushed by StarPU in a Modularized Scheduler, the task moves from a Scheduling Component to another, following the hierarchy of the Scheduling Tree, and is stored in one of the Scheduling Components of the strategy. When a worker wants to pop a task from the Modularized Scheduler, the corresponding Worker Component of the Scheduling Tree tries to pull a task from its parents, following the hierarchy, and gives it to the worker if it succeeded to get one.

38.4.1 Interface

Each Scheduling Component must follow the following pre-defined Interface to be able to interact with other Scheduling Components.

- push_task (child_component, Task)
  The calling Scheduling Component transfers a task to its Child Component. When the Push function returns, the task no longer belongs to the calling Component. The Modularized Schedulers' model relies on this function to perform prefetching. See starpu_sched_component::push_task for more details
- pull_task (parent_component, caller_component) -> Task
  The calling Scheduling Component requests a task from its Parent Component. When the Pull function ends, the returned task belongs to the calling Component. See starpu_sched_component::pull_task for more details
- can_push (caller_component, parent_component)
  The calling Scheduling Component notifies its Parent Component that it is ready to accept new tasks. See starpu_sched_component::can_push for more details
- can_pull (caller_component, child_component)
  The calling Scheduling Component notifies its Child Component that it is ready to give new tasks. See starpu_sched_component::can_pull for more details

The components also provide the following useful methods:

- starpu_sched_component::estimated_load provides an estimated load of the component

- starpu_sched_component::estimated_end provides an estimated date of availability of workers behind the component, after processing tasks in the component and below. This is computed only if the estimated field of the tasks have been set before passing it to the component.

38.4.2 Building a Modularized Scheduler

38.4.2.1 Pre-implemented Components

StarPU is currently shipped with the following four Scheduling Components :

- Storage Components : Fifo, Prio
  Components which store tasks. They can also prioritize them if they have a defined priority. It is possible to define a threshold for those Components following two criteria : the number of tasks stored in the Component, or the sum of the expected length of all tasks stored in the Component. When a push operation tries to queue a task beyond the threshold, the push fails. When some task leaves the queue (and thus possibly more tasks can fit), this component calls can_push from ancestors.
- Resource-Mapping Components : Mct, Heft, Eager, Random, Work-Stealing
  "Core" of the Scheduling Strategy, those Components are the ones who make scheduling choices between their children components.
- Worker Components : Worker
  Each Worker Component modelizes a concrete worker, and copes with the technical tricks of interacting with the StarPU core. Modular schedulers thus usually have them at the bottom of their component tree.
- Special-Purpose Components : Perfmodel_Select, Best_Implementation
  Components dedicated to original purposes. The Perfmodel_Select Component decides which Resource-Mapping Component should be used to schedule a task: a component that assumes tasks with a calibrated performance model; a component for non-yet-calibrated tasks, that will distribute them to get measurements done as quickly as possible; and a component that takes the tasks without performance models.
  The Best_Implementation Component chooses which implementation of a task should be used on the chosen resource.

38.4.2.2 Progression And Validation Rules

Some rules must be followed to ensure the correctness of a Modularized Scheduler :

- At least one Storage Component without threshold is needed in a Modularized Scheduler, to store incoming tasks from StarPU. It can for instance be a global component at the top of the tree, or one component per worker at the bottom of the tree, or intermediate assemblies. The important point is that the starpu_sched_component::push_task call at the top can not fail, so there has to be a storage component without threshold between the top of the tree and the first storage component with threshold, or the workers themselves.
- At least one Resource-Mapping Component is needed in a Modularized Scheduler. Resource-Mapping Components are the only ones which can make scheduling choices, and so the only ones which can have several children.

38.4.2.3 Locking in modularized schedulers

Most often, components do not need to take locks. This allows e.g. the push operation to be called in parallel when tasks get released in parallel from different workers which have completed different ancestor tasks.
When a component has internal information which needs to be kept coherent, the component can define its own lock to take it as it sees fit, e.g. to protect a task queue. This may however limit scalability of the scheduler. Conversely, since push and pull operations will be called concurrently from different workers, the component might prefer to use a central mutex to serialize all scheduling decisions to avoid pathological cases (all push calls decide to put their task on the same target)

38.4.2.4 Implementing a Modularized Scheduler

The following code shows how to implement a Tree-Eager-Prefetching Scheduler.

```
static void initialize_eager_prefetching_center_policy(unsigned sched_ctx_id)
{
  /* The eager component will decide for each task which worker will run it,
* and we want fifos both above and below the component */
  starpu_sched_component_initialize_simple_scheduler(
    starpu_sched_component_eager_create, NULL,
    STARPU_SCHED_SIMPLE_DECIDE_WORKERS |
    STARPU_SCHED_SIMPLE_FIFO_ABOVE |
    STARPU_SCHED_SIMPLE_FIFOS_BELOW,
    sched_ctx_id);
}
/* Initializing the starpu_sched_policy struct associated to the Modularized
* Scheduler :  only the init_sched and deinit_sched needs to be defined to
* implement a Modularized Scheduler */
struct starpu_sched_policy _starpu_sched_tree_eager_prefetching_policy =
{
  .init_sched = initialize_eager_prefetching_center_policy,
  .deinit_sched = starpu_sched_tree_deinitialize,
  .add_workers = starpu_sched_tree_add_workers,
  .remove_workers = starpu_sched_tree_remove_workers,
  .push_task = starpu_sched_tree_push_task,
  .pop_task = starpu_sched_tree_pop_task,
  .pre_exec_hook = starpu_sched_component_worker_pre_exec_hook,
  .post_exec_hook = starpu_sched_component_worker_post_exec_hook,
  .policy_name = "tree-eager-prefetching",
  .policy_description = "eager with prefetching tree policy"
};
```

starpu_sched_component_initialize_simple_scheduler() is a helper function which makes it very trivial to assemble a modular scheduler around a scheduling decision component as seen above (here, a dumb eager decision component). Most often, a modular scheduler can be implemented that way.

A modular scheduler can also be constructed hierarchically with starpu_sched_component_composed_recipe_create().

To retrieve the current scheduling tree of a task, starpu_sched_tree_get() can be called.

That modular scheduler can also be built by hand in the following way:

```
#define _STARPU_SCHED_NTASKS_THRESHOLD_DEFAULT 2
#define _STARPU_SCHED_EXP_LEN_THRESHOLD_DEFAULT 1000000000.0
static void initialize_eager_prefetching_center_policy(unsigned sched_ctx_id)
{
  unsigned ntasks_threshold = _STARPU_SCHED_NTASKS_THRESHOLD_DEFAULT;
  double exp_len_threshold = _STARPU_SCHED_EXP_LEN_THRESHOLD_DEFAULT;
  [...]
  starpu_sched_ctx_create_worker_collection
    (sched_ctx_id, STARPU_WORKER_LIST);
  /* Create the Scheduling Tree */
  struct starpu_sched_tree * t = starpu_sched_tree_create(sched_ctx_id);
  /* The Root Component is a Flow-control Fifo Component */
   t->root = starpu_sched_component_fifo_create(NULL);
  /* The Resource-mapping Component of the strategy is an Eager Component
*/
  struct starpu_sched_component *eager_component = starpu_sched_component_eager_create(NULL);
  /* Create links between Components :  the Eager Component is the child
* of the Root Component */
  starpu_sched_component_connect(t->root, eager_component);
  /* A task threshold is set for the Flow-control Components which will
* be connected to Worker Components.  By doing so, this Modularized
* Scheduler will be able to perform some prefetching on the resources
*/
  struct starpu_sched_component_fifo_data fifo_data =
  {
    .ntasks_threshold = ntasks_threshold,
    .exp_len_threshold = exp_len_threshold,
  };
  unsigned i;
  for(i = 0; i < starpu_worker_get_count() + starpu_combined_worker_get_count(); i++)
  {
    /* Each Worker Component has a Flow-control Fifo Component as
* father */
    struct starpu_sched_component * worker_component = starpu_sched_component_worker_new(i);
    struct starpu_sched_component * fifo_component = starpu_sched_component_fifo_create(&fifo_data);
    starpu_sched_component_connect(fifo_component, worker_component);
    /* Each Flow-control Fifo Component associated to a Worker
* Component is linked to the Eager Component as one of its
* children */
    starpu_sched_component_connect(eager_component, fifo_component);
  }
  starpu_sched_tree_update_workers(t);
  starpu_sched_ctx_set_policy_data(sched_ctx_id, (void*)t);
}
/* Properly destroy the Scheduling Tree and all its Components */
static void deinitialize_eager_prefetching_center_policy(unsigned sched_ctx_id)
{
  struct starpu_sched_tree * tree = (struct
      starpu_sched_tree*)starpu_sched_ctx_get_policy_data(sched_ctx_id);
  starpu_sched_tree_destroy(tree);
  starpu_sched_ctx_delete_worker_collection(sched_ctx_id);
}
/* Initializing the starpu_sched_policy struct associated to the Modularized
* Scheduler :  only the init_sched and deinit_sched needs to be defined to
* implement a Modularized Scheduler */
struct starpu_sched_policy _starpu_sched_tree_eager_prefetching_policy =
{
  .init_sched = initialize_eager_prefetching_center_policy,
  .deinit_sched = deinitialize_eager_prefetching_center_policy,
  .add_workers = starpu_sched_tree_add_workers,
  .remove_workers = starpu_sched_tree_remove_workers,
  .push_task = starpu_sched_tree_push_task,
  .pop_task = starpu_sched_tree_pop_task,
  .pre_exec_hook = starpu_sched_component_worker_pre_exec_hook,
  .post_exec_hook = starpu_sched_component_worker_post_exec_hook,
  .policy_name = "tree-eager-prefetching",
  .policy_description = "eager with prefetching tree policy"
};
```

Instead of calling starpu_sched_tree_update_workers(), one can call starpu_sched_tree_update_workers_in_ctx() to update the set of workers that are available to execute tasks in a given scheduling tree within a specific StarPU context.

Other modular scheduler examples can be seen in `src/sched_policies/modular_*.c`

For instance, `modular-heft-prio` needs performance models, decides memory nodes, uses prioritized fifos above and below, and decides the best implementation.

If unsure on the result of the modular scheduler construction, you can run a simple application with FxT enabled (see Generating Traces With FxT), and open the generated file `trace.html` in a web-browser.

38.4.3 Management of parallel task

At the moment, parallel tasks can be managed in modularized schedulers through combined workers: instead of connecting a scheduling component to a worker component, one can connect it to a combined worker component (i.e. a worker component created with a combined worker id). That component will handle creating task aliases for parallel execution and push them to the different workers components.

38.4.4 Writing a Scheduling Component

38.4.4.1 Generic Scheduling Component

Each Scheduling Component is instantiated from a Generic Scheduling Component, which implements a generic version of the Interface. The generic implementation of Pull, Can_Pull and Can_Push functions are recursive calls to their parents (respectively to their children). However, as a Generic Scheduling Component do not know how many children it will have when it will be instantiated, it does not implement the Push function.

38.4.4.2 Instantiation : Redefining the Interface

A Scheduling Component must implement all the functions of the Interface. It is so necessary to implement a Push function to instantiate a Scheduling Component. The implemented Push function is the "fingerprint" of a Scheduling Component. Depending on how functionalities or properties programmers want to give to the Scheduling Component they are implementing, it is possible to reimplement all the functions of the Interface. For example, a Flow-control Component reimplements the Pull and the Can_Push functions of the Interface, allowing to catch the generic recursive calls of these functions. The Pull function of a Flow-control Component can, for example, pop a task from the local storage queue of the Component, and give it to the calling Component which asks for it.

38.4.4.3 Detailed Progression and Validation Rules

- A Reservoir is a Scheduling Component which redefines a Push and a Pull function, in order to store tasks into it. A Reservoir delimit Scheduling Areas in the Scheduling Tree.
- A Pump is the engine source of the Scheduler : it pushes/pulls tasks to/from a Scheduling Component to another. Native Pumps of a Scheduling Tree are located at the root of the Tree (incoming Push calls from StarPU), and at the leafs of the Tree (Pop calls coming from StarPU Workers). Pre-implemented Scheduling Components currently shipped with Pumps are Flow-Control Components and the Resource-Mapping Component Heft, within their defined Can_Push functions.

- A correct Scheduling Tree requires a Pump per Scheduling Area and per Execution Flow.

The Tree-Eager-Prefetching Scheduler shown in Section Implementing a Modularized Scheduler follows the previous assumptions :

```
                                 starpu_push_task
                                      Pump
                                        |
Area 1                                  |
                                        |
                                        v
           -----------------------Fifo_Component-----------------------------
                                      Pump
                                       |  ^
                               Push    |  |    Can_Push
                                       v  |
Area 2                           Eager_Component
                                       |  ^
                                       |  |
                                       v  |
                     --------><-------------------><---------
                     |  ^                                |  ^
             Push    |  |    Can_Push            Push    |  |    Can_Push
                     v  |                                v  |
           -----Fifo_Component-----------------------Fifo_Component----------
                     |  ^                                |  ^
             Pull    |  |    Can_Pull            Pull    |  |    Can_Pull
Area 3               v  |                                v  |
                    Pump                               Pump
               Worker_Component                    Worker_Component
```

38.5 Using a New Scheduling Policy

There are two ways to use a new scheduling policy.

- If the code is directly available from your application, you can set the field starpu_conf::sched_policy with a pointer to your new defined scheduling policy.
```
starpu_conf_init(&conf);
conf.sched_policy = &dummy_sched_policy,
ret = starpu_init(&conf);
```
- You can also load the new policy dynamically using the environment variable STARPU_SCHED_LIB. An example is given in `examples/scheduler/libdummy_sched.c` and `examples/scheduler/libdummy_sched.sh`.

  The variable STARPU_SCHED_LIB needs to give the location of a .so file which needs to define a function `struct starpu_sched_policy *starpu_get_sched_lib_policy(const char *name)`
```
struct starpu_sched_policy *get_sched_policy(const char *name)
{
        if (!strcmp(name, "dummy"))
                return &dummy_sched_policy;
        return NULL;
}
```
  To use it, you need to define both variables STARPU_SCHED_LIB and STARPU_SCHED
```
STARPU_SCHED_LIB=libdummy_sched.so STARPU_SCHED=dummy yourapplication
```
  If the library defines a function `struct starpu_sched_policy **starpu_get_sched_lib_policies()`, the policies defined by the library can be displayed using the `help` functionality.
```
STARPU_SCHED_LIB=libdummy_sched.so STARPU_SCHED=help yourapplication
```

38.6 Graph-based Scheduling

For performance reasons, most of the schedulers shipped with StarPU use simple list-scheduling heuristics, assuming that the application has already set priorities. This is why they do their scheduling between when tasks become available for execution and when a worker becomes idle, without looking at the task graph.

Other heuristics can however look at the task graph. Recording the task graph is expensive, so it is not available by default, the scheduling heuristic has to set `_starpu_graph_record` to `1` from the initialization function, to make it available. Then the `_starpu_graph*` functions can be used.
`src/sched_policies/graph_test_policy.c` is an example of simple greedy policy which automatically computes priorities by bottom-up rank.
The idea is that while the application submits tasks, they are only pushed to a bag of tasks. When the application is finished with submitting tasks, it calls starpu_do_schedule() (or starpu_task_wait_for_all(), which calls starpu_do_schedule()), and the starpu_sched_policy::do_schedule method of the scheduler is called. This method calls `_starpu_graph_compute_depths()` to compute the bottom-up ranks, and then uses these ranks to set priorities over tasks.
It then has two priority queues, one for CPUs, and one for GPUs, and uses a dumb heuristic based on the duration of the task over CPUs and GPUs to decide between the two queues. CPU workers can then pop from the CPU priority queue, and GPU workers from the GPU priority queue.

38.7 Debugging Scheduling

All the Online Performance Tools and Offline Performance Tools can be used to get information about how well the execution proceeded, and thus the overall quality of the execution.
Precise debugging can also be performed by using the STARPU_TASK_BREAK_ON_PUSH, STARPU_TASK_BREAK_ON_SCHED, STARPU_TASK_BREAK_ON_POP, and STARPU_TASK_BREAK_ON_EXEC environment variables. By setting the job_id of a task in these environment variables, StarPU will raise `SIGTRAP` when the task is being scheduled, pushed, or popped by the scheduler. This means that when one notices that a task is being scheduled in a seemingly odd way, one can just re-execute the application in a debugger, with some of those variables set, and the execution will stop exactly at the scheduling points of this task, thus allowing to inspect the scheduler state, etc.

Chapter 39

CUDA Support

StarPU sets the current CUDA device by calling starpu_cuda_set_device() which takes an integer argument representing the device number, and sets the current device to the specified device number. By setting the current device, applications can select which CUDA device to use for their computations, enabling efficient management of multiple CUDA devices in a system.
We can call starpu_cuda_get_nvmldev() to get identifier of the NVML device associated with a given CUDA device.
Three macros STARPU_CUDA_REPORT_ERROR(), STARPU_CUBLAS_REPORT_ERROR(), and STARPU_CUSPARSE_REPORT_ERROR() are useful for debugging and troubleshooting, as they provide detailed information about the error that occur during CUDA or CUBLAS execution.

Chapter 40

OpenCL Support

StarPU provides several functions for managing OpenCL programs and kernels. starpu_opencl_load_program_source() and starpu_opencl_load_program_source_malloc() load the OpenCL program source from a file, but the latter one also allocates buffer for the program source. starpu_opencl_compile_opencl_from_file() and starpu_opencl_compile_opencl_from_string() are used to compile an OpenCL kernel from a source file or a string respectively. starpu_opencl_load_binary_opencl() is used to compile the binary OpenCL kernel. An example is available in `examples/binary/binary.c`.

starpu_opencl_load_opencl_from_file() and starpu_opencl_load_opencl_from_string() are used to compile an OpenCL source code from a file or a string respectively. starpu_opencl_unload_opencl() is used to unload an OpenCL compiled program or kernel from memory. starpu_opencl_load_opencl() is used to create an OpenCL kernel for specified device. starpu_opencl_release_kernel() is used to release the specified OpenCL kernel. An example illustrating the usage of OpenCL support is available in `examples/basic_examples/vector_scal_opencl.c`.

For managing OpenCL contexts, devices, and command queues, there are several functions: starpu_opencl_get_context(), starpu_opencl_get_device() and starpu_opencl_get_queue() are used to retrieve the OpenCL context, device and command queue associated with a given device number respectively. starpu_opencl_get_current_context() and starpu_opencl_get_current_queue() are used to retrieve the OpenCL context or command queue of the current worker that is being used by the calling thread. We can call starpu_opencl_set_kernel_args() to set the arguments for an OpenCL kernel. Examples are available in `examples/filters/custom_mf/`.

Two functions are useful for debugging and error reporting in OpenCL applications. starpu_opencl_error_string() takes an OpenCL error code as an argument and returns a string containing a description of the error. starpu_opencl_display_error() takes an OpenCL error code as an argument and prints the corresponding error message to the standard error stream.

Chapter 41

Maxeler FPGA Support

41.1 Introduction

Maxeler provides hardware and software solutions for accelerating computing applications on dataflow engines (DFEs). DFEs are in-house designed accelerators that encapsulate reconfigurable high-end FPGAs at their core and are equipped with large amounts of DDR memory.
We extend the StarPU task programming library that initially targets heterogeneous architectures to support Field Programmable Gate Array (FPGA).
To create `StarPU/FPGA` applications exploiting DFE configurations, MaxCompiler allows an application to be split into three parts:

- `Kernel`, which implements the computational components of the application in hardware.
- `Manager configuration`, which connects Kernels to the CPU, engine RAM, other Kernels and other DFEs via MaxRing.
- `CPU application`, which interacts with the DFEs to read and write data to the Kernels and engine RAM.

The Simple Live CPU interface (SLiC) is Maxeler's application programming interface for seamless CPU-DFE integration. SLiC allows CPU applications to configure and load a number of DFEs as well as to subsequently schedule and run actions on those DFEs using simple function calls. In StarPU/FPGA applications, we use *Dynamic SLiC Interface* to exchange data streams between the CPU (Main Memory) and DFE (Local Memory).

41.2 Porting Applications to Maxeler FPGA

The way to port an application to FPGA is to set the field starpu_codelet::max_fpga_funcs, to provide StarPU with the function for FPGA implementation, so for instance:

```
struct starpu_codelet cl =
{
    .max_fpga_funcs = {myfunc},
    .nbuffers = 1,
}
```

A basic example is available in the file `tests/maxfpga/max_fpga_basic_static.c`.

41.2.1 StarPU/Maxeler FPGA Application

To give you an idea of the interface that we used to exchange data between `host` (CPU) and `FPGA` (DFE), here is an example, based on one of the examples of Maxeler (`https://trac.version.fz-juelich.de/reconfigurable/wiki/Public`).
`StreamFMAKernel.maxj` represents the Java kernel code; it implements a very simple kernel (`c=a+b`), and `Test.c` starts it from the `fpga_add` function; it first sets streaming up from the CPU pointers, triggers execution and waits for the result. The API to interact with DFEs is called *SLiC* which then also involves the `MaxelerOS` runtime.

- `StreamFMAKernel.maxj`: the DFE part is described in the MaxJ programming language, which is a Java-based metaprogramming approach.

```
package tests;
import com.maxeler.maxcompiler.v2.kernelcompiler.Kernel;
import com.maxeler.maxcompiler.v2.kernelcompiler.KernelParameters;
import com.maxeler.maxcompiler.v2.kernelcompiler.types.base.DFEType;
import com.maxeler.maxcompiler.v2.kernelcompiler.types.base.DFEVar;
class StreamFMAKernel extends Kernel
{
   private static final DFEType type = dfeInt(32);
   protected StreamFMAKernel(KernelParameters parameters)
   {
             super(parameters);
             DFEVar a = io.input("a", type);
             DFEVar b = io.input("b", type);
             DFEVar c;
             c = a+b;
             io.output("output", c, type);
   }
}
```

- `StreamFMAManager.maxj`: is also described in the MaxJ programming language and orchestrates data movement between the host and the DFE.

```
package tests;
import com.maxeler.maxcompiler.v2.build.EngineParameters;
import com.maxeler.maxcompiler.v2.managers.custom.blocks.KernelBlock;
import com.maxeler.platform.max5.manager.Max5LimaManager;
class StreamFMAManager extends Max5LimaManager
{
        private static final String kernel_name = "StreamFMAKernel";
        public StreamFMAManager(EngineParameters arg0)
        {
                super(arg0);
                KernelBlock kernel = addKernel(new StreamFMAKernel(makeKernelParameters(kernel_name)));
                kernel.getInput("a") <== addStreamFromCPU("a");
                kernel.getInput("b") <== addStreamFromCPU("b");
                addStreamToCPU("output") <== kernel.getOutput("output");
        }
        public static void main(String[] args)
        {
                StreamFMAManager manager = new StreamFMAManager(new EngineParameters(args));
                manager.build();
        }
}
```

Once `StreamFMAKernel.maxj` and `StreamFMAManager.maxj` are written, there are other steps to do:

- Building the JAVA program: (for Kernel and Manager (.maxj))

  ```
  $ maxjc -1.7 -cp $MAXCLASSPATH streamfma/
  ```

- Running the Java program to generate a DFE implementation (a .max file) that can be called from a StarPU/FPGA application and slic headers (.h) for simulation:

```
$ java -XX:+UseSerialGC -Xmx2048m -cp $MAXCLASSPATH:. streamfma.StreamFMAManager DFEModel=MAIA maxFileName=Str
```

- Build the slic object file (simulation):

```
$ sliccompile StreamFMA.max
```

- `Test.c :`

to interface StarPU task-based runtime system with Maxeler's DFE devices, we use the advanced dynamic interface of *SLiC* in **non_blocking** mode.

Test code must include `MaxSLiCInterface.h` and `MaxFile.h`. The .max file contains the bitstream. The StarPU/FPGA application can be written in C, C++, etc. Some examples are available in the directory `tests/maxfpga`.

```
#include "StreamFMA.h"
#include "MaxSLiCInterface.h"
void fpga_add(void *buffers[], void *cl_arg)
{
    (void)cl_arg;
    int *a = (int*) STARPU_VECTOR_GET_PTR(buffers[0]);
    int *b = (int*) STARPU_VECTOR_GET_PTR(buffers[1]);
    int *c = (int*) STARPU_VECTOR_GET_PTR(buffers[2]);
    int size = STARPU_VECTOR_GET_NX(buffers[0]);
    /* actions to run on an engine */
```

```
    max_actions_t *act = max_actions_init(maxfile, NULL);
    /* set the number of ticks for a kernel */
    max_set_ticks  (act, "StreamFMAKernel", size);
    /* send input streams */
    max_queue_input(act, "a", a, size *sizeof(a[0]));
    max_queue_input(act, "b", b, size*sizeof(b[0]));
    /* store output stream */
    max_queue_output(act,"output", c, size*sizeof(c[0]));
    /* run actions on the engine */
     printf("**** Run actions in non blocking mode **** \n");
    /* run actions in non_blocking mode */
    max_run_t *run0= max_run_nonblock(engine, act);
    printf("*** wait for the actions on DFE to complete *** \n");
    max_wait(run0);
  }
  static struct starpu_codelet cl =
  {
    .cpu_funcs = {cpu_func},
    .cpu_funcs_name = {"cpu_func"},
    .max_fpga_funcs = {fpga_add},
    .nbuffers = 3,
    .modes = {STARPU_R, STARPU_R, STARPU_W}
  };
int main(int argc, char **argv)
{
    ...
    /* Implementation of a maxfile */
    max_file_t *maxfile = StreamFMA_init();
    /* Implementation of an engine */
    max_engine_t *engine = max_load(maxfile, "*");
    starpu_init(NULL);
    ...  Task submission etc.  ...
    starpu_shutdown();
    /* deallocate the set of actions */
    max_actions_free(act);
    /* unload and deallocate an engine obtained by way of max_load */
    max_unload(engine);
    return 0;
}
```

To write the StarPU/FPGA application: first, the programmer must describe the codelet using StarPU's C API. This codelet provides both a CPU implementation and an FPGA one. It also specifies that the task has two inputs and one output through the starpu_codelet::nbuffers and starpu_codelet::modes attributes.
`fpga_add` function is the name of the FPGA implementation and is mainly divided in four steps:

- Init actions to be run on DFE.
- Add data to an input stream for an action.
- Add data storage space for an output stream.
- Run actions on DFE in **non_blocking** mode; a non-blocking call returns immediately, allowing the calling code to do more CPU work in parallel while the actions are run.
- Wait for the actions to complete.

In the `main` function, there are four important steps:

- Implement a maxfile.
- Load a DFE.
- Free actions.
- Unload and deallocate the DFE.

The rest of the application (data registration, task submission, etc.) is as usual with StarPU.
The design load can also be delegated to StarPU by specifying an array of load specifications in `starpu_conf::max_fpga_load`, and use starpu_max_fpga_get_local_engine() to access the loaded max engines.
Complete examples are available in `tests/fpga/*.c`

41.2.2 Data Transfers in StarPU/Maxeler FPGA Applications

The communication between the host and the DFE is done through the *Dynamic advance interface* to exchange data between the main memory and the local memory of the DFE.
For the moment, we use STARPU_MAIN_RAM to send and store data to/from DFE's local memory. However, we aim to use a multiplexer to choose which memory node we will use to read/write data. So, users can tell that the computational kernel will take data from the main memory or DFE's local memory, for example.
In StarPU applications, when starpu_codelet::specific_nodes is set to 1, this specifies the memory nodes where each data should be sent to for task execution.

41.2.3 Maxeler FPGA Configuration

To configure StarPU with Maxeler FPGA accelerators, make sure that the `slic-config` is available from your `PATH` environment variable.

41.2.4 Launching Programs: Simulation

Maxeler provides a simple tutorial to use MaxCompiler (`https://trac.version.fz-juelich.de/reconfigurable/wiki/Public`). Running the Java program to generate maxfile and slic headers (hardware) on Maxeler's DFE device, takes a VERY long time, approx. 2 hours even for this very small example. That's why we use the simulation.

- To start the simulation on Maxeler's DFE device:

```
$ maxcompilersim -c LIMA -n StreamFMA restart
```

- To run the binary (simulation)

```
$ export LD_LIBRARY_PATH=$MAXELEROSDIR/lib:$LD_LIBRARY_PATH
$ export SLIC_CONF="use_simulation=StreamFMA"
```

- To force tasks to be scheduled on the FPGA, one can disable the use of CPU cores by setting the STARPU_NCPU environment variable to 0.

```
$ STARPU_NCPU=0 ./StreamFMA
```

- To stop the simulation

```
$ maxcompilersim -c LIMA -n StreamFMA stop
```

Chapter 42

Out Of Core

42.1 Introduction

When using StarPU, one may need to store more data than what the main memory (RAM) can store. This part describes the method to add a new memory node on a disk and to use it.

Similarly to what happens with GPUs (it's actually exactly the same code), when available main memory becomes scarce, StarPU will evict unused data to the disk, thus leaving room for new allocations. Whenever some evicted data is needed again for a task, StarPU will automatically fetch it back from the disk.

The principle is that one first registers a disk memory node with a set of functions to manipulate data by calling starpu_disk_register(), and then registers a disk location, seen by StarPU as a `void*`, which can be for instance a Unix path for the `stdio`, `unistd` or `unistd_o_direct` backends, or a leveldb database for the `leveldb` backend, an HDF5 file path for the `HDF5` backend, etc. The `disk` backend opens this place with the plug() method. StarPU can then start using it to allocate room and store data there with the disk write method, without user intervention.

Users can also use starpu_disk_open() to explicitly open an object within the disk, e.g. a file name in the `stdio` or `unistd` cases, or a database key in the `leveldb` case, and then use `starpu_*_register` functions to turn it into a StarPU data handle. StarPU will then use this file as an external source of data, and automatically read and write data as appropriate. In the end use starpu_disk_close() to close an existing object.

In any case, users also need to set STARPU_LIMIT_CPU_MEM to the amount of data that StarPU will be allowed to afford. By default, StarPU will use the machine memory size, but part of it is taken by the kernel, the system, daemons, and the application's own allocated data, whose size can not be predicted. That is why users need to specify what StarPU can afford.

Some Out-of-core tests are worth giving a read, see `tests/disk/*.c`

42.2 Use a new disk memory

To use a disk memory node, you have to register it with this function:

```
int new_dd = starpu_disk_register(&starpu_disk_unistd_ops, (void *) "/tmp/", 1024*1024*200);
```

Here, we use the `unistd` library to realize the read/write operations, i.e. `fread/fwrite`. This structure must have a path where to store files, as well as the maximum size the software can afford to store on the disk.

Don't forget to check if the result is correct!

This can also be achieved by just setting environment variables STARPU_DISK_SWAP, STARPU_DISK_SWAP_BACKEND and STARPU_DISK_SWAP_SIZE :

```
export STARPU_DISK_SWAP=/tmp
export STARPU_DISK_SWAP_BACKEND=unistd
export STARPU_DISK_SWAP_SIZE=200
```

The backend can be set to `stdio` (some caching is done by `libc` and the kernel), `unistd` (only caching in the kernel), `unistd_o_direct` (no caching), `leveldb`, or `hdf5`.

It is important to understand that when the backend is not set to `unistd_o_direct`, some caching will occur at the kernel level (the page cache), which will also consume memory... STARPU_LIMIT_CPU_MEM might need to be set to less than half of the machine memory just to leave room for the kernel's page cache, otherwise the kernel will struggle to get memory. Using `unistd_o_direct` avoids this caching, thus allowing to set STARPU_LIMIT_CPU_MEM to the machine memory size (minus some memory for normal kernel operations, system daemons, and application data).

When the register call is made, StarPU will benchmark the disk. This can take some time.
Warning: the size thus has to be at least STARPU_DISK_SIZE_MIN bytes !
StarPU will then automatically try to evict unused data to this new disk. One can also use the standard StarPU memory node API to prefetch data etc., see the Standard Memory Library and the Data Interfaces.
The disk is unregistered during the execution of starpu_shutdown().

42.3 Data Registration

StarPU will only be able to achieve Out-Of-Core eviction if it controls memory allocation. For instance, if the application does the following:

```
p = malloc(1024*1024*sizeof(float));
fill_with_data(p);
starpu_matrix_data_register(&h, STARPU_MAIN_RAM, (uintptr_t) p, 1024, 1024, 1024, sizeof(float));
```

StarPU will not be able to release the corresponding memory since it's the application which allocated it, and StarPU can not know how, and thus how to release it. One thus have to use the following instead:

```
starpu_matrix_data_register(&h, -1, NULL, 1024, 1024, 1024, sizeof(float));
starpu_task_insert(cl_fill_with_data, STARPU_W, h, 0);
```

Which makes StarPU automatically do the allocation when the task running cl_fill_with_data gets executed. And then if it needs to, it will be able to release it after having pushed the data to the disk. Since no initial buffer is provided to starpu_matrix_data_register(), the handle does not have any initial value right after this call, and thus the very first task using the handle needs to use the STARPU_W mode like above, STARPU_R or STARPU_RW would not make sense.
By default, StarPU will try to push any data handle to the disk. To specify whether a given handle should be pushed to the disk, starpu_data_set_ooc_flag() should be used. To get to know whether a given handle should be pushed to the disk, starpu_data_get_ooc_flag() should be used.

42.4 Using Wont Use

By default, StarPU uses a Least-Recently-Used (LRU) algorithm to determine which data should be evicted to the disk. This algorithm can be hinted by telling which data will not be used in the coming future thanks to starpu_data_wont_use(), for instance:

```
starpu_task_insert(&cl_work, STARPU_RW, h, 0);
starpu_data_wont_use(h);
```

StarPU will mark the data as "inactive" and tend to evict to the disk that data rather than others.

42.5 Examples: disk_copy

```
/* Try to write into disk memory
* Use mechanism to push data from main ram to disk ram
*/
#include <starpu.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
/* size of one vector */
#define NX      (30*1000000/sizeof(double))
#define FPRINTF(ofile, fmt, ...)  do { if (!getenv("STARPU_SSILENT")) {fprintf(ofile, fmt, ## __VA_ARGS__);
      }} while(0)
int main(int argc, char **argv)
{
        double *A, *F;
        /* limit main ram to force to push in disk */
        setenv("STARPU_LIMIT_CPU_MEM", "160", 1);
        /* Initialize StarPU with default configuration */
        int ret = starpu_init(NULL);
        if (ret == -ENODEV) goto enodev;
        /* register a disk */
        int new_dd = starpu_disk_register(&starpu_disk_unistd_ops, (void *) "/tmp/", 1024*1024*200);
        /* can't write on /tmp/ */
        if (new_dd == -ENOENT) goto enoent;
        /* allocate two memory spaces */
        starpu_malloc_flags((void **)&A, NX*sizeof(double), STARPU_MALLOC_COUNT);
        starpu_malloc_flags((void **)&F, NX*sizeof(double), STARPU_MALLOC_COUNT);
        FPRINTF(stderr, "TEST DISK MEMORY \n");
        unsigned int j;
        /* initialization with bad values */
        for(j = 0; j < NX; ++j)
        {
                A[j] = j;
                F[j] = -j;
        }
        starpu_data_handle_t vector_handleA, vector_handleB, vector_handleC, vector_handleD, vector_handleE,
      vector_handleF;
        /* register vector in starpu */
        starpu_vector_data_register(&vector_handleA, STARPU_MAIN_RAM, (uintptr_t)A, NX, sizeof(double));
        starpu_vector_data_register(&vector_handleB, -1, (uintptr_t) NULL, NX, sizeof(double));
        starpu_vector_data_register(&vector_handleC, -1, (uintptr_t) NULL, NX, sizeof(double));
        starpu_vector_data_register(&vector_handleD, -1, (uintptr_t) NULL, NX, sizeof(double));
        starpu_vector_data_register(&vector_handleE, -1, (uintptr_t) NULL, NX, sizeof(double));
        starpu_vector_data_register(&vector_handleF, STARPU_MAIN_RAM, (uintptr_t)F, NX, sizeof(double));
        /* copy vector A->B, B->C...  */
        starpu_data_cpy(vector_handleB, vector_handleA, 0, NULL, NULL);
        starpu_data_cpy(vector_handleC, vector_handleB, 0, NULL, NULL);
        starpu_data_cpy(vector_handleD, vector_handleC, 0, NULL, NULL);
        starpu_data_cpy(vector_handleE, vector_handleD, 0, NULL, NULL);
        starpu_data_cpy(vector_handleF, vector_handleE, 0, NULL, NULL);
        /* StarPU does not need to manipulate the array anymore so we can stop
* monitoring it */
        /* free them */
        starpu_data_unregister(vector_handleA);
        starpu_data_unregister(vector_handleB);
        starpu_data_unregister(vector_handleC);
        starpu_data_unregister(vector_handleD);
        starpu_data_unregister(vector_handleE);
        starpu_data_unregister(vector_handleF);
        /* check if computation is correct */
        int try = 1;
        for (j = 0; j < NX; ++j)
                if (A[j] != F[j])
                {
                        printf("Fail A %f != F %f \n", A[j], F[j]);
                        try = 0;
                }
        /* free last vectors */
        starpu_free_flags(A, NX*sizeof(double), STARPU_MALLOC_COUNT);
        starpu_free_flags(F, NX*sizeof(double), STARPU_MALLOC_COUNT);
        /* terminate StarPU, no task can be submitted after */
        starpu_shutdown();
        if(try)
                FPRINTF(stderr, "TEST SUCCESS\n");
        else
                FPRINTF(stderr, "TEST FAIL\n");
        return (try ?  EXIT_SUCCESS : EXIT_FAILURE);
enodev:
        return 77;
enoent:
        return 77;
}
```

The full code is provided in the file `tests/disk/disk_copy.c`

42.6 Examples: disk_compute

```
/* Try to write into disk memory
* Use mechanism to push data from main ram to disk ram
*/
#include <starpu.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <math.h>
#define NX (1024)
int main(int argc, char **argv)
{
        /* Initialize StarPU with default configuration */
        int ret = starpu_init(NULL);
        if (ret == -ENODEV) goto enodev;
        /* Initialize path and name */
        char pid_str[16];
        int pid = getpid();
        snprintf(pid_str, sizeof(pid_str), "%d", pid);
        const char *name_file_start = "STARPU_DISK_COMPUTE_DATA_";
        const char *name_file_end = "STARPU_DISK_COMPUTE_DATA_RESULT_";
        char * path_file_start = malloc(strlen(base) + 1 + strlen(name_file_start) + 1);
        strcpy(path_file_start, base);
        strcat(path_file_start, "/");
        strcat(path_file_start, name_file_start);
        char * path_file_end = malloc(strlen(base) + 1 + strlen(name_file_end) + 1);
        strcpy(path_file_end, base);
        strcat(path_file_end, "/");
        strcat(path_file_end, name_file_end);
        /* register a disk */
        int new_dd = starpu_disk_register(&starpu_disk_unistd_ops, (void *) base, 1024*1024*1);
        /* can't write on /tmp/ */
        if (new_dd == -ENOENT) goto enoent;
        unsigned dd = (unsigned) new_dd;
        printf("TEST DISK MEMORY \n");
        /* Imagine, you want to compute data */
        int *A;
        int *C;
        starpu_malloc_flags((void **)&A, NX*sizeof(int), STARPU_MALLOC_COUNT);
        starpu_malloc_flags((void **)&C, NX*sizeof(int), STARPU_MALLOC_COUNT);
        unsigned int j;
        /* you register them in a vector */
        for(j = 0; j < NX; ++j)
        {
                A[j] = j;
                C[j] = 0;
        }
        /* you create a file to store the vector ON the disk */
        FILE * f = fopen(path_file_start, "wb+");
        if (f == NULL)
                goto enoent2;
        /* store it in the file */
        fwrite(A, sizeof(int), NX, f);
        /* close the file */
        fclose(f);
        /* create a file to store result */
        f = fopen(path_file_end, "wb+");
        if (f == NULL)
                goto enoent2;
        /* replace all data by 0 */
        fwrite(C, sizeof(int), NX, f);
        /* close the file */
        fclose(f);
        /* And now, you want to use your data in StarPU */
        /* Open the file ON the disk */
        void * data = starpu_disk_open(dd, (void *) name_file_start, NX*sizeof(int));
        void * data_result = starpu_disk_open(dd, (void *) name_file_end, NX*sizeof(int));
        starpu_data_handle_t vector_handleA, vector_handleC;
        /* register vector in starpu */
        starpu_vector_data_register(&vector_handleA, dd, (uintptr_t) data, NX, sizeof(int));
        /* and do what you want with it, here we copy it into an other vector */
        starpu_vector_data_register(&vector_handleC, dd, (uintptr_t) data_result, NX, sizeof(int));
        starpu_data_cpy(vector_handleC, vector_handleA, 0, NULL, NULL);
        /* free them */
        starpu_data_unregister(vector_handleA);
        starpu_data_unregister(vector_handleC);
        /* close them in StarPU */
        starpu_disk_close(dd, data, NX*sizeof(int));
        starpu_disk_close(dd, data_result, NX*sizeof(int));
        /* check results */
        f = fopen(path_file_end, "rb+");
        if (f == NULL)
                goto enoent;
        /* take data */
        fread(C, sizeof(int), NX, f);
        /* close the file */
        fclose(f);
        int try = 1;
        for (j = 0; j < NX; ++j)
                if (A[j] != C[j])
                {
                        printf("Fail A %d != C %d \n", A[j], C[j]);
                        try = 0;
                }
        starpu_free_flags(A, NX*sizeof(int), STARPU_MALLOC_COUNT);
        starpu_free_flags(C, NX*sizeof(int), STARPU_MALLOC_COUNT);
        unlink(path_file_start);
        unlink(path_file_end);
        free(path_file_start);
        free(path_file_end);
        /* terminate StarPU, no task can be submitted after */
        starpu_shutdown();
        if(try)
                printf("TEST SUCCESS\n");
        else
                printf("TEST FAIL\n");
        return (try ?  EXIT_SUCCESS : EXIT_FAILURE);
enodev:
        return 77;
enoent2:
        starpu_free_flags(A, NX*sizeof(int), STARPU_MALLOC_COUNT);
        starpu_free_flags(C, NX*sizeof(int), STARPU_MALLOC_COUNT);
enoent:
        unlink(path_file_start);
        unlink(path_file_end);
        free(path_file_start);
        free(path_file_end);
        starpu_shutdown();
        return 77;
}
```

The full code is provided in the file `tests/disk/disk_compute.c`

42.7 Performances

Scheduling heuristics for Out-of-core are still relatively experimental. The tricky part is that you usually have to find a compromise between privileging locality (which avoids back and forth with the disk) and privileging the critical path, i.e. taking into account priorities to avoid lack of parallelism at the end of the task graph.
It is notably better to avoid defining different priorities to tasks with low priority, since that will make the scheduler want to schedule them by levels of priority, at the expense of locality.
The scheduling algorithms worth trying are thus `dmdar` and `lws`, which privilege data locality over priorities. There will be work on this area in the coming future.

42.8 Feedback Figures

Beyond pure performance feedback, some figures are interesting to have a look at.
Using `export STARPU_BUS_STATS=1` (STARPU_BUS_STATS and STARPU_BUS_STATS_FILE to define a filename in which to display statistics, by default the standard error stream is used) gives an overview of the data transfers which were needed. The values can also be obtained at runtime by using starpu_bus_get_profiling_info(). An example can be read in `src/profiling/profiling_helpers.c`.

```
#---------------------
Data transfer speed for /tmp/sthibault-disk-DJzhAj (node 1):
0 -> 1: 99 MB/s
1 -> 0: 99 MB/s
0 -> 1: 23858 µs
1 -> 0: 23858 µs

#---------------------
TEST DISK MEMORY

#---------------------
Data transfer stats:
        Disk 0 -> NUMA 0        0.0000 GB       0.0000 MB/s     (transfers : 0 - avg -nan MB)
        NUMA 0 -> Disk 0        0.0625 GB       63.6816 MB/s    (transfers : 2 - avg 32.0000 MB)
Total transfers: 0.0625 GB
#---------------------
```

Using `export STARPU_ENABLE_STATS=1` gives information for each memory node on data miss/hit and allocation miss/hit.

```
#---------------------
MSI cache stats :
memory node NUMA 0
        hit : 32 (66.67 %)
        miss : 16 (33.33 %)
memory node Disk 0
        hit : 0 (0.00 %)
        miss : 0 (0.00 %)
#---------------------

#---------------------
Allocation cache stats:
memory node NUMA 0
        total alloc : 16
        cached alloc: 0 (0.00 %)
memory node Disk 0
        total alloc : 8
        cached alloc: 0 (0.00 %)
#---------------------
```

42.9 Disk functions

There are various ways to operate a disk memory node, described by the structure starpu_disk_ops. For instance, the variable starpu_disk_unistd_ops uses read/write functions.

All structures are in Out Of Core.
Examples are provided in `src/core/disk_ops/disk_*.c`

Chapter 43

MPI Support

The integration of MPI transfers within task parallelism is done in a very natural way by the means of asynchronous interactions between the application and StarPU. This is implemented in a separate `libstarpumpi` library which basically provides "StarPU" equivalents of `MPI_*` functions, where `void *` buffers are replaced with starpu_data_handle_t, and all GPU-RAM-NIC transfers are handled efficiently by StarPU-MPI. Users have to use the usual `mpirun` command of the MPI implementation to start StarPU on the different MPI nodes.
An MPI Insert Task function provides an even more seamless transition to a distributed application, by automatically issuing all required data transfers according to the task graph and an application-provided distribution.
Some source codes are available in the directory `mpi/`.

43.1 Building with MPI support

If a `mpicc` compiler is already in your PATH, StarPU will automatically enable MPI support in the build. If `mpicc` is not in PATH, you can specify its location by passing `-with-mpicc=/where/there/is/mpicc` to `./configure`
It can be useful to enable MPI tests during `make check` by passing `-enable-mpi-check` to `./configure`. And similarly to `mpicc`, if `mpiexec` in not in PATH, you can specify its location by passing `-with-mpiexec=/where/there/is/mpiexec` to `./configure`, but this is not needed if it is next to `mpicc`, configure will look there in addition to PATH.
Similarly, Fortran examples use `mpif90`, which can be specified manually with `-with-mpifort` if it can't be found automatically.
If users want to run several MPI processes by machine (e.g. one per NUMA node), STARPU_WORKERS_GETBIND needs to be left to its default value 1 to make StarPU take into account the binding set by the MPI launcher (otherwise each StarPU instance would try to bind on all cores of the machine...)
However, depending on the architecture of your machine, one may end up with StarPU-MPI nodes not having any CPU workers. If a node only gets 1 CPU, it will be bound to the MPI thread, and none will be left to start a CPU worker.
One can check that with the following commands.

```
$ mpirun -np 2 starpu_machine_display --worker CPU --count --notopology
1 CPU worker
1 CPU worker
$ mpirun -np 4 starpu_machine_display --worker CPU --count --notopology
4 CPU workers
4 CPU workers
4 CPU workers
4 CPU workers
$ mpirun --bind-to socket -np 2 starpu_machine_display --worker CPU --count --notopology
4 CPU workers
4 CPU workers
$ STARPU_WORKERS_GETBIND=0 mpirun -np 4 starpu_machine_display --worker CPU --count --notopology
4 CPU workers
4 CPU workers
4 CPU workers
4 CPU workers
$ STARPU_WORKERS_GETBIND=0 mpirun -np 2 starpu_machine_display --worker CPU --count --notopology
4 CPU workers
4 CPU workers
```

or with `hwloc`

```
mpirun --bind-to socket -np 2 hwloc-ls --restrict binding --no-io
mpirun -np 2 hwloc-ls --restrict binding --no-io
```

43.2 Example Used In This Documentation

The example below will be used as the base for this documentation. It initializes a token on node 0, and the token is passed from node to node, incremented by one on each step. The code is not using StarPU yet.

```
for (loop = 0; loop < nloops; loop++)
{
    int tag = loop*size + rank;
    if (loop == 0 && rank == 0)
    {
        token = 0;
        fprintf(stdout, "Start with token value %d\n", token);
    }
    else
    {
        MPI_Recv(&token, 1, MPI_INT, (rank+size-1)%size, tag, MPI_COMM_WORLD);
    }
    token++;
    if (loop == last_loop && rank == last_rank)
    {
        fprintf(stdout, "Finished:  token value %d\n", token);
    }
    else
    {
        MPI_Send(&token, 1, MPI_INT, (rank+1)%size, tag+1, MPI_COMM_WORLD);
    }
}
```

43.3 About Not Using The MPI Support

Although StarPU provides MPI support, the application programmer may want to keep his MPI communications as they are for a start, and only delegate task execution to StarPU. This is possible by just using starpu_data_acquire(), for instance:

```
for (loop = 0; loop < nloops; loop++)
{
    int tag = loop*size + rank;
    /* Acquire the data to be able to write to it */
    starpu_data_acquire(token_handle, STARPU_W);
    if (loop == 0 && rank == 0)
    {
        token = 0;
        fprintf(stdout, "Start with token value %d\n", token);
    }
    else
    {
        MPI_Recv(&token, 1, MPI_INT, (rank+size-1)%size, tag, MPI_COMM_WORLD);
    }
    starpu_data_release(token_handle);
    /* Task delegation to StarPU to increment the token.  The execution might
* be performed on a CPU, a GPU, etc.  */
    increment_token();
    /* Acquire the update data to be able to read from it */
    starpu_data_acquire(token_handle, STARPU_R);
    if (loop == last_loop && rank == last_rank)
    {
        fprintf(stdout, "Finished:  token value %d\n", token);
    }
    else
    {
        MPI_Send(&token, 1, MPI_INT, (rank+1)%size, tag+1, MPI_COMM_WORLD);
    }
    starpu_data_release(token_handle);
}
```

In that case, `libstarpumpi` is not needed. One can also use `MPI_Isend()` and `MPI_Irecv()`, by calling starpu_data_release() after `MPI_Wait()` or `MPI_Test()` have notified completion.

It is however better to use `libstarpumpi`, to save the application from having to synchronize with starpu_data_acquire(), and instead just submit all tasks and communications asynchronously, and wait for the overall completion.

43.4 Simple Example

The flags required to compile or link against the MPI layer are accessible with the following commands:

```
$ pkg-config --cflags starpumpi-1.4  # options for the compiler
$ pkg-config --libs starpumpi-1.4    # options for the linker
```

```
void increment_token(void)
{
    struct starpu_task *task = starpu_task_create();
    task->cl = &increment_cl;
    task->handles[0] = token_handle;
    starpu_task_submit(task);
}
int main(int argc, char **argv)
{
    int rank, size;
    starpu_mpi_init_conf(&argc, &argv, 1, MPI_COMM_WORLD, NULL);
    starpu_mpi_comm_rank(MPI_COMM_WORLD, &rank);
    starpu_mpi_comm_size(MPI_COMM_WORLD, &size);
    starpu_vector_data_register(&token_handle, STARPU_MAIN_RAM, (uintptr_t)&token, 1, sizeof(unsigned));
    unsigned nloops = NITER;
    unsigned loop;
    unsigned last_loop = nloops - 1;
    unsigned last_rank = size - 1;
    for (loop = 0; loop < nloops; loop++)
    {
        int tag = loop*size + rank;
        if (loop == 0 && rank == 0)
        {
            starpu_data_acquire(token_handle, STARPU_W);
            token = 0;
            fprintf(stdout, "Start with token value %d\n", token);
            starpu_data_release(token_handle);
        }
        else
        {
            starpu_mpi_irecv_detached(token_handle, (rank+size-1)%size, tag, MPI_COMM_WORLD, NULL, NULL);
        }
        increment_token();
        if (loop == last_loop && rank == last_rank)
        {
            starpu_data_acquire(token_handle, STARPU_R);
            fprintf(stdout, "Finished:  token value %d\n", token);
            starpu_data_release(token_handle);
        }
        else
        {
            starpu_mpi_isend_detached(token_handle, (rank+1)%size, tag+1, MPI_COMM_WORLD, NULL, NULL);
        }
    }
    starpu_task_wait_for_all();
    starpu_mpi_shutdown();
    if (rank == last_rank)
    {
        fprintf(stderr, "[%d] token = %d == %d * %d ?\n", rank, token, nloops, size);
        STARPU_ASSERT(token == nloops*size);
    }
```

We have here replaced `MPI_Recv()` and `MPI_Send()` with starpu_mpi_irecv_detached() and starpu_mpi_isend_detached(), which just submit the communication to be performed. The implicit sequential consistency dependencies provide synchronization between MPI reception and emission and the corresponding tasks. The only remaining synchronization with starpu_data_acquire() is at the beginning and the end.
The full source code is available in the file `mpi/tests/ring.c`.

43.5 How to Initialize StarPU-MPI

As seen in the previous example, one has to call starpu_mpi_init_conf() to initialize StarPU-MPI. The third parameter of the function indicates if MPI should be initialized by StarPU, or if the application did it itself. If the application initializes MPI itself, it must call `MPI_Init_thread()` with `MPI_THREAD_SERIALIZED` or `MPI_THREAD_MULTIPLE`, since StarPU-MPI uses a separate thread to perform the communications. `MPI_THREAD_MULTIPLE` is necessary if the application also performs some MPI communications.

43.6 Point To Point Communication

The standard point to point communications of MPI have been implemented. The semantic is similar to the MPI one, but adapted to the DSM provided by StarPU. An MPI request will only be submitted when the data is available in the main memory of the node submitting the request.

There are two types of asynchronous communications: the classic asynchronous communications and the detached communications. The classic asynchronous communications (starpu_mpi_isend() and starpu_mpi_irecv()) need to be followed by a call to starpu_mpi_wait() or to starpu_mpi_test() to wait for or to test the completion of the communication. As shown in the example `mpi/tests/async_ring.c`. Waiting for or testing the completion of detached communications is not possible, this is done internally by StarPU-MPI, on completion, the resources are automatically released. This mechanism is similar to the pthread detach state attribute, which determines whether a thread will be created in a joinable or a detached state.

For send communications, data is acquired with the mode STARPU_R. When using the `configure` option --enable-mpi-pedantic-isend, the mode STARPU_RW is used to make sure there is no more than 1 concurrent `MPI_Isend()` call accessing a data and StarPU does not read from it from tasks during the communication.

Internally, all communication are divided in 2 communications, a first message is used to exchange an envelope describing the data (i.e. its tag and its size), the data itself is sent in a second message. All MPI communications submitted by StarPU uses a unique tag, which has a default value. This value can be accessed with the function starpu_mpi_get_communication_tag() and changed with the function starpu_mpi_set_communication_tag(). The matching of tags with corresponding requests is done within StarPU-MPI.

For any userland communication, the call of the corresponding function (e.g. starpu_mpi_isend()) will result in the creation of a StarPU-MPI request, the function starpu_data_acquire_cb() is then called to asynchronously request StarPU to fetch the data in main memory; when the data is ready and the corresponding buffer has already been received by MPI, it will be copied in the memory of the data, otherwise the request is stored in the *early requests list*. Sending requests are stored in the *ready requests list*.

While requests need to be processed, the StarPU-MPI progression thread does the following:

1. it polls the *ready requests list*. For all the ready requests, the appropriate function is called to post the corresponding MPI call. For example, an initial call to starpu_mpi_isend() will result in a call to `MPI_Isend()`. If the request is marked as detached, the request will then be added to the *detached requests list*.

2. it posts an `MPI_Irecv()` to retrieve a data envelope.

3. it polls the *detached requests list*. For all the detached requests, it tests its completion of the MPI request by calling `MPI_Test()`. On completion, the data handle is released, and if a callback was defined, it is called.

4. finally, it checks if a data envelope has been received. If so, if the data envelope matches a request in the *early requests list* (i.e. the request has already been posted by the application), the corresponding MPI call is posted (similarly to the first step above).

   If the data envelope does not match any application request, a temporary handle is created to receive the data, a StarPU-MPI request is created and added into the *ready requests list*, and thus will be processed in the first step of the next loop.

To prevent putting too much pressure on the MPI library, only a limited number of requests are emitted concurrently. This behavior can be tuned with the environment variable STARPU_MPI_NDETACHED_SEND. In the same fashion, the progression thread will poll for termination of existing requests after submitting a defined number of requests. This behavior can be tuned with the environment variable STARPU_MPI_NREADY_PROCESS.

The function starpu_mpi_issend() allows to perform a synchronous-mode, non-blocking send of a data. It can also be specified when using starpu_mpi_task_insert() with the parameter STARPU_SSEND.

MPIPtpCommunication gives the list of all the point to point communications defined in StarPU-MPI.

43.7 Exchanging User Defined Data Interface

New data interfaces defined as explained in Defining A New Data Interface can also be used within StarPU-MPI and exchanged between nodes. Two functions needs to be defined through the type starpu_data_interface_ops. The function starpu_data_interface_ops::pack_data takes a handle and returns a contiguous memory buffer allocated with

```
starpu_malloc_flags(ptr, size, 0)
```

along with its size, where data to be conveyed to another node should be copied.

```
static int complex_pack_data(starpu_data_handle_t handle, unsigned node, void **ptr, ssize_t *count)
{
    STARPU_ASSERT(starpu_data_test_if_allocated_on_node(handle, node));
    struct starpu_complex_interface *complex_interface = (struct starpu_complex_interface *)
      starpu_data_get_interface_on_node(handle, node);
    *count = complex_get_size(handle);
    *ptr = starpu_malloc_on_node_flags(node, *count, 0);
    memcpy(*ptr, complex_interface->real, complex_interface->nx*sizeof(double));
    memcpy(*ptr+complex_interface->nx*sizeof(double), complex_interface->imaginary,
      complex_interface->nx*sizeof(double));
    return 0;
}
```

The inverse operation is implemented in the function starpu_data_interface_ops::unpack_data which takes a contiguous memory buffer and recreates the data handle.

```
static int complex_unpack_data(starpu_data_handle_t handle, unsigned node, void *ptr, size_t count)
{
    STARPU_ASSERT(starpu_data_test_if_allocated_on_node(handle, node));
    struct starpu_complex_interface *complex_interface = (struct starpu_complex_interface *)
      starpu_data_get_interface_on_node(handle, node);
    memcpy(complex_interface->real, ptr, complex_interface->nx*sizeof(double));
    memcpy(complex_interface->imaginary, ptr+complex_interface->nx*sizeof(double),
      complex_interface->nx*sizeof(double));
    starpu_free_on_node_flags(node, (uintptr_t) ptr, count, 0);
    return 0;
}
```

And the starpu_data_interface_ops::peek_data operation does the same, but without freeing the buffer. Of course, one can implement starpu_data_interface_ops::unpack_data as merely calling starpu_data_interface_ops::peek_data and do the free:

```
static int complex_peek_data(starpu_data_handle_t handle, unsigned node, void *ptr, size_t count)
{
    STARPU_ASSERT(starpu_data_test_if_allocated_on_node(handle, node));
    STARPU_ASSERT(count == complex_get_size(handle));
    struct starpu_complex_interface *complex_interface = (struct starpu_complex_interface *)
      starpu_data_get_interface_on_node(handle, node);
    memcpy(complex_interface->real, ptr, complex_interface->nx*sizeof(double));
    memcpy(complex_interface->imaginary, ptr+complex_interface->nx*sizeof(double),
      complex_interface->nx*sizeof(double));
    return 0;
}
static struct starpu_data_interface_ops interface_complex_ops =
{
    ...
    .pack_data = complex_pack_data,
    .peek_data = complex_peek_data
    .unpack_data = complex_unpack_data
};
```

Instead of defining pack and unpack operations, users may want to attach an MPI type to their user-defined data interface. The function starpu_mpi_interface_datatype_register() allows doing so. This function takes 3 parameters: the interface ID for which the MPI datatype is going to be defined, a function's pointer that will create the MPI datatype, and a function's pointer that will free the MPI datatype. If for some data an MPI datatype can not be built (e.g. complex data structure), the creation function can return -1, StarPU-MPI will then fallback to using pack/unpack.

The functions to create and free the MPI datatype are defined and registered as follows.

```
void starpu_complex_interface_datatype_allocate(starpu_data_handle_t handle, MPI_Datatype *mpi_datatype)
{
    int ret;
    int blocklengths[2];
    MPI_Aint displacements[2];
    MPI_Datatype types[2] = {MPI_DOUBLE, MPI_DOUBLE};
    struct starpu_complex_interface *complex_interface = (struct starpu_complex_interface *)
      starpu_data_get_interface_on_node(handle, STARPU_MAIN_RAM);
    MPI_Get_address(complex_interface, displacements);
    MPI_Get_address(&complex_interface->imaginary, displacements+1);
    displacements[1] -= displacements[0];
    displacements[0] = 0;
    blocklengths[0] = complex_interface->nx;
    blocklengths[1] = complex_interface->nx;
    ret = MPI_Type_create_struct(2, blocklengths, displacements, types, mpi_datatype);
    STARPU_ASSERT_MSG(ret == MPI_SUCCESS, "MPI_Type_contiguous failed");
    ret = MPI_Type_commit(mpi_datatype);
    STARPU_ASSERT_MSG(ret == MPI_SUCCESS, "MPI_Type_commit failed");
}
void starpu_complex_interface_datatype_free(MPI_Datatype *mpi_datatype)
{
    MPI_Type_free(mpi_datatype);
}
static struct starpu_data_interface_ops interface_complex_ops =
{
    ...
};
interface_complex_ops.interfaceid = starpu_data_interface_get_next_id();
starpu_mpi_interface_datatype_register(interface_complex_ops.interfaceid,
      starpu_complex_interface_datatype_allocate, starpu_complex_interface_datatype_free);
starpu_data_interface handle;
starpu_complex_data_register(&handle, STARPU_MAIN_RAM, real, imaginary, 2);
...
```

An example is provided in the file `mpi/examples/user_datatype/my_interface.c`.
It is also possible to use starpu_mpi_datatype_register() to register the functions through a handle rather than the interface ID, but note that in that case it is important to make sure no communication is going to occur before the function starpu_mpi_datatype_register() is called. This would otherwise produce an undefined result as the data may be received before the function is called, and so the MPI datatype would not be known by the StarPU-MPI communication engine, and the data would be processed with the pack and unpack operations. One would thus need to synchronize all nodes:

```
starpu_data_interface handle;
starpu_complex_data_register(&handle, STARPU_MAIN_RAM, real, imaginary, 2);
starpu_mpi_datatype_register(handle, starpu_complex_interface_datatype_allocate,
      starpu_complex_interface_datatype_free);
starpu_mpi_barrier(MPI_COMM_WORLD);
```

43.8 MPI Insert Task Utility

To save the programmer from having to specify all communications, StarPU provides an "MPI Insert Task Utility". The principle is that the application decides a distribution of the data over the MPI nodes by allocating it and notifying StarPU of this decision, i.e. tell StarPU which MPI node "owns" which data. It also decides, for each handle, an MPI tag which will be used to exchange the content of the handle. All MPI nodes then process the whole task graph, and StarPU automatically determines which node actually execute which task, and trigger the required MPI transfers.
The list of functions is described in MPIInsertTask.
Here is an stencil example showing how to use starpu_mpi_task_insert(). One first needs to define a distribution function which specifies the locality of the data. Note that the data needs to be registered to MPI by calling starpu_mpi_data_register(). This function allows setting the distribution information and the MPI tag which should be used when communicating the data. It also allows to automatically clear the MPI communication cache when unregistering the data. A basic example is in the file `mpi/tests/insert_task.c`.

```
/* Returns the MPI node number where data is */
int my_distrib(int x, int y, int nb_nodes)
{
  /* Block distrib */
  return ((int)(x / sqrt(nb_nodes) + (y / sqrt(nb_nodes)) * sqrt(nb_nodes))) % nb_nodes;
  // /* Other examples useful for other kinds of computations */
  // /* / distrib */
  // return (x+y) % nb_nodes;
  // /* Block cyclic distrib */
  // unsigned side = sqrt(nb_nodes);
  // return x % side + (y % side) * size;
}
```

Now the data can be registered within StarPU. Data which are not owned but will be needed for computations can be registered through the lazy allocation mechanism, i.e. with a `home_node` set to `-1`. StarPU will automatically allocate the memory when it is used for the first time.
One can note an optimization here (the `else if` test): we only register data which will be needed by the tasks that we will execute.

```
unsigned matrix[X][Y];
starpu_data_handle_t data_handles[X][Y];
for(x = 0; x < X; x++)
{
    for (y = 0; y < Y; y++)
    {
        int mpi_rank = my_distrib(x, y, size);
        if (mpi_rank == my_rank)
            /* Owning data */
            starpu_variable_data_register(&data_handles[x][y], STARPU_MAIN_RAM, (uintptr_t)&(matrix[x][y]),
      sizeof(unsigned));
        else if (my_rank == my_distrib(x+1, y, size) || my_rank == my_distrib(x-1, y, size)
              || my_rank == my_distrib(x, y+1, size) || my_rank == my_distrib(x, y-1, size))
            /* I don't own this index, but will need it for my computations */
            starpu_variable_data_register(&data_handles[x][y], -1, (uintptr_t)NULL, sizeof(unsigned));
        else
            /* I know it's useless to allocate anything for this */
            data_handles[x][y] = NULL;
        if (data_handles[x][y])
        {
            starpu_mpi_data_register(data_handles[x][y], x*X+y, mpi_rank);
        }
    }
}
```

Now starpu_mpi_task_insert() can be called for the different steps of the application.

```
for(loop=0 ; loop<niter; loop++)
    for (x = 1; x < X-1; x++)
        for (y = 1; y < Y-1; y++)
            starpu_mpi_task_insert(MPI_COMM_WORLD, &stencil5_cl,
                                   STARPU_RW, data_handles[x][y],
                                   STARPU_R, data_handles[x-1][y],
                                   STARPU_R, data_handles[x+1][y],
                                   STARPU_R, data_handles[x][y-1],
                                   STARPU_R, data_handles[x][y+1],
                                   0);
starpu_task_wait_for_all();
```

The full source code is available in the file `mpi/examples/stencil/stencil5.c`.

I.e. all MPI nodes process the whole task graph, but as mentioned above, for each task, only the MPI node which owns the data being written to (here, `data_handles[x][y]`) will actually run the task. The other MPI nodes will automatically send the required data.

To tune the placement of tasks among MPI nodes, one can use STARPU_EXECUTE_ON_NODE or STARPU_EXECUTE_ON_DATA to specify an explicit node (an example can be found in `mpi/tests/insert_task_node_choice.c`), or the node of a given data (e.g. one of the parameters), or use starpu_mpi_node_selection_register_poli and STARPU_NODE_SELECTION_POLICY to provide a dynamic policy (an example can be found in `mpi/tests/policy_register.c`). The default policy is to execute the task on the node which owns a data that require write access; if the task requires several data handles with write access, the node executing the task is selected in order to minimize the amount of data to transfer between nodes.

A function starpu_mpi_task_build() is also provided with the aim to only construct the task structure. All MPI nodes need to call the function, which posts the required send/recv on the various nodes as needed. Only the node which is to execute the task will then return a valid task structure, others will return `NULL`. This node must submit the task. All nodes then need to call the function starpu_mpi_task_post_build() – with the same list of arguments as starpu_mpi_task_build() – to post all the necessary data communications meant to happen after the task execution.

```
struct starpu_task *task;
task = starpu_mpi_task_build(MPI_COMM_WORLD, &cl,
                             STARPU_RW, data_handles[0],
                             STARPU_R, data_handles[1],
                             0);
if (task) starpu_task_submit(task);
starpu_mpi_task_post_build(MPI_COMM_WORLD, &cl,
                           STARPU_RW, data_handles[0],
                           STARPU_R, data_handles[1],
                           0);
```

A full source code using these functions is available in the file `mpi/tests/insert_task_compute.c`.

It is also possible to create and submit the task outside of StarPU-MPI functions and call the functions starpu_mpi_task_exchange_data_before_execution() and starpu_mpi_task_exchange_data_after_execution() to exchange data as required by the data ownership's nodes.

```
struct starpu_mpi_task_exchange_params params;
struct starpu_data_descr descrs[2];
struct starpu_task *task;
task = starpu_task_create();
task->cl = &mycodelet;
task->handles[0] = data_handles[0];
task->handles[1] = data_handles[1];
starpu_mpi_task_exchange_data_before_execution(MPI_COMM_WORLD, task, descrs, &params);
if (params.do_execute) starpu_task_submit(task);
starpu_mpi_task_exchange_data_after_execution(MPI_COMM_WORLD, descrs, 2, params);
```

A full source code using these functions is available in the file `mpi/tests/mpi_task_submit.c`.

If many data handles must be registered with unique tag ids, or if multiple applications are concurrently submitting tasks to StarPU, it is then difficult to keep the uniqueness of the tags for each piece of data. StarPU provides a tag management system to allocate/free a unique range of tags when registering the data to prevent conflict from one application to another. The previous code then becomes:

```
unsigned matrix[X][Y];
starpu_data_handle_t data_handles[X][Y];
int64_t mintag = starpu_mpi_tags_allocate(X*Y);
for(x = 0; x < X; x++)
{
    for (y = 0; y < Y; y++)
    {
            ...
        if (data_handles[x][y])
        {
            starpu_mpi_data_register(data_handles[x][y], mintag + y*Y+x, mpi_rank);
        }
    }
}
```

Then, when all these pieces of data have been unregistered, you may free the range of tags by calling:

```
starpu_mpi_tags_free(mintag);
```

where `mintag` was the value returned by starpu_mpi_tags_allocate().

Note that both these functions should be called by all nodes involved in the computations in the exact same order and with the same parameters to keep the tags synchronized between all nodes.

Also note that StarPU will not check if a tag given to starpu_mpi_data_register() has been previously registered, this functionality only aims to prevent different parts of an application to use the same data tags.

43.9 Other MPI Utility Functions

Similarly to the function starpu_data_cpy(), the function starpu_mpi_data_cpy() can be used to transfer a data between 2 nodes. It behaves as starpu_data_cpy() if both data are owned by the same node, otherwise a transfer is initiated between the nodes. A priority and a callback function can be defined.

```
...
starpu_mpi_data_register(src_handle, 12, 0); // Data is owned by node0
starpu_mpi_data_register(dst_handle, 42, 1); // Data is owned by node1
...
starpu_mpi_data_cpy(dst_handle, src_handle, MPI_COMM_WORLD, 0, callback, NULL);
```

43.10 Pruning MPI Task Insertion

Making all MPI nodes process the whole graph can be a concern with a growing number of nodes. To avoid this, the application can prune the task for loops according to the data distribution, to only submit tasks on nodes which have to care about them (either to execute them, or to send the required data).

A way to do some of this quite easily can be to just add an `if` like this:

```
for(loop=0 ; loop<niter; loop++)
    for (x = 1; x < X-1; x++)
        for (y = 1; y < Y-1; y++)
            if (my_distrib(x,y,size) == my_rank
             || my_distrib(x-1,y,size) == my_rank
             || my_distrib(x+1,y,size) == my_rank
             || my_distrib(x,y-1,size) == my_rank
             || my_distrib(x,y+1,size) == my_rank)
                starpu_mpi_task_insert(MPI_COMM_WORLD, &stencil5_cl,
                                       STARPU_RW, data_handles[x][y],
                                       STARPU_R, data_handles[x-1][y],
                                       STARPU_R, data_handles[x+1][y],
                                       STARPU_R, data_handles[x][y-1],
                                       STARPU_R, data_handles[x][y+1],
                                       0);
starpu_task_wait_for_all();
```

This permits to drop the cost of function call argument passing and parsing.

An example is available in the file `examples/stencil/implicit-stencil-tasks.c`.

If the `my_distrib` function can be inlined by the compiler, the latter can improve the test.

If the `size` can be made a compile-time constant, the compiler can considerably improve the test further.

If the distribution function is not too complex and the compiler is very good, the latter can even optimize the `for` loops, thus dramatically reducing the cost of task submission.

To estimate quickly how long task submission takes, and notably how much pruning saves, a quick and easy way is to measure the submission time of just one of the MPI nodes. This can be achieved by running the application on just one MPI node with the following environment variables:

```
export STARPU_DISABLE_KERNELS=1
export STARPU_MPI_FAKE_RANK=2
export STARPU_MPI_FAKE_SIZE=1024
```

Here we have disabled the kernel function call to skip the actual computation time and only keep submission time, and we have asked StarPU to fake running on MPI node 2 out of 1024 nodes.

43.11 Temporary Data

To be able to use starpu_mpi_task_insert(), one has to call starpu_mpi_data_register(), so that StarPU-MPI can know what it needs to do for each data. Parameters of starpu_mpi_data_register() are normally the same on all nodes for a given data, so that all nodes agree on which node owns the data, and which tag is used to transfer its value.

It can however be useful to register e.g. some temporary data on just one node, without having to register a dumb handle on all nodes, while only one node will actually need to know about it. In this case, nodes which will not need the data can just pass `NULL` to starpu_mpi_task_insert():

```
starpu_data_handle_t data0 = NULL;
if (rank == 0)
{
    starpu_variable_data_register(&data0, STARPU_MAIN_RAM, (uintptr_t) &val0, sizeof(val0));
    starpu_mpi_data_register(data0, 0, rank);
}
starpu_mpi_task_insert(MPI_COMM_WORLD, &cl, STARPU_W, data0, 0); /* Executes on node 0 */
```

Here, nodes whose rank is not `0` will simply not take care of the data, and consider it to be on another node.

This can be mixed various way, for instance here node `1` determines that it does not have to care about `data0`, but knows that it should send the value of its `data1` to node `0`, which owns data and thus will need the value of `data1` to execute the task:

```
starpu_data_handle_t data0 = NULL, data1, data;
if (rank == 0)
{
    starpu_variable_data_register(&data0, STARPU_MAIN_RAM, (uintptr_t) &val0, sizeof(val0));
    starpu_mpi_data_register(data0, -1, rank);
    starpu_variable_data_register(&data1, -1, 0, sizeof(val1));
    starpu_variable_data_register(&data, STARPU_MAIN_RAM, (uintptr_t) &val, sizeof(val));
}
else if (rank == 1)
{
    starpu_variable_data_register(&data1, STARPU_MAIN_RAM, (uintptr_t) &val1, sizeof(val1));
    starpu_variable_data_register(&data, -1, 0, sizeof(val));
}
starpu_mpi_data_register(data, 42, 0);
starpu_mpi_data_register(data1, 43, 1);
starpu_mpi_task_insert(MPI_COMM_WORLD, &cl, STARPU_W, data, STARPU_R, data0, STARPU_R, data1, 0); /*
      Executes on node 0 */
```

The full source code is available in the file `mpi/tests/temporary.c`.

43.12 Per-node Data

Further than temporary data on just one node, one may want per-node data, to e.g. replicate some computation because that is less expensive than communicating the value over MPI:

```
starpu_data_handle pernode, data0, data1;
starpu_variable_data_register(&pernode, -1, 0, sizeof(val));
starpu_mpi_data_register(pernode, -1, STARPU_MPI_PER_NODE);
/* Normal data:  one on node0, one on node1 */
if (rank == 0)
{
    starpu_variable_data_register(&data0, STARPU_MAIN_RAM, (uintptr_t) &val0, sizeof(val0));
    starpu_variable_data_register(&data1, -1, 0, sizeof(val1));
}
else if (rank == 1)
{
    starpu_variable_data_register(&data0, -1, 0, sizeof(val1));
    starpu_variable_data_register(&data1, STARPU_MAIN_RAM, (uintptr_t) &val1, sizeof(val1));
}
starpu_mpi_data_register(data0, 42, 0);
starpu_mpi_data_register(data1, 43, 1);
starpu_mpi_task_insert(MPI_COMM_WORLD, &cl, STARPU_W, pernode, 0); /* Will be replicated on all nodes */
starpu_mpi_task_insert(MPI_COMM_WORLD, &cl2, STARPU_RW, data0, STARPU_R, pernode); /* Will execute on node
      0, using its own pernode*/
starpu_mpi_task_insert(MPI_COMM_WORLD, &cl2, STARPU_RW, data1, STARPU_R, pernode); /* Will execute on node
      1, using its own pernode*/
```

One can turn a normal data into per-node data, by first broadcasting it to all nodes:

```
starpu_data_handle data;
starpu_variable_data_register(&data, -1, 0, sizeof(val));
starpu_mpi_data_register(data, 42, 0);
/* Compute some value */
starpu_mpi_task_insert(MPI_COMM_WORLD, &cl, STARPU_W, data, 0); /* Node 0 computes it */
/* Get it on all nodes */
starpu_mpi_get_data_on_all_nodes_detached(MPI_COMM_WORLD, data);
/* And turn it per-node */
starpu_mpi_data_set_rank(data, STARPU_MPI_PER_NODE);
```

The data can then be used just like per-node above.

The full source code is available in the file `mpi/tests/temporary.c`.

43.13 Inter-node reduction

One might want to leverage a reduction pattern across several nodes. Using STARPU_REDUX (see Data Reduction), one can obtain such patterns where each core on contributing nodes spawns their own copy to work with. In the case that the required reductions are too numerous and expensive, the access mode STARPU_MPI_REDUX tells StarPU to spawn only one contribution per contributing node.

The setup and use of STARPU_MPI_REDUX is similar to STARPU_REDUX : the initialization and reduction codelets should be declared through starpu_data_set_reduction_methods() in the same fashion as STARPU_REDUX. Example `mpi/examples/mpi_redux/mpi_redux.c` shows how to use the STARPU_MPI_REDUX mode and compare it with the standard STARPU_REDUX. The function starpu_mpi_redux_data() is automatically called either when a task reading the reduced handle is inserted through the MPI layer of StarPU through starpu_mpi_insert_task() or when users wait for all communications and tasks to be executed through starpu_mpi_wait_for_all(). The function can be called by users to fine-tune arguments such as the priority of the reduction tasks. Tasks contributing to the inter-node reduction should be registered as accessing the contribution through STARPU_RW|STARPU_COMMUTE mode, as for the STARPU_REDUX mode, as in the following example.

```
static struct starpu_codelet contrib_cl =
{
    .cpu_funcs = {cpu_contrib}, /* cpu implementation(s) of the routine */
    .nbuffers = 1, /* number of data handles referenced by this routine */
    .modes = {STARPU_RW | STARPU_COMMUTE} /* access modes for the contribution */
    .name = "contribution"
};
```

When inserting these tasks, the access mode handed out to the StarPU-MPI layer should be `STARPU_MPI_REDUX`. If a task uses a `data` owned by node 0 and is executed on the node 1, it can be inserted as in the following example.

```
starpu_mpi_task_insert(MPI_COMM_WORLD, &contrib_cl, STARPU_MPI_REDUX, data, STARPU_EXECUTE_ON_NODE, 1); /*
      Node 1 computes it */
```

Note that if the specified node is set to `-1`, the option is ignored.

More examples are available at `mpi/examples/mpi_redux/mpi_redux.c` and `mpi/examples/mpi_redux/mpi_redux_tree.c`.

43.14 Priorities

All send functions have a `_prio` variant which takes an additional priority parameter, which allows making StarPU-MPI change the order of MPI requests before submitting them to MPI. The default priority is `0`.

When using the starpu_mpi_task_insert() helper, STARPU_PRIORITY defines both the task priority and the MPI requests priority. An example is available in the file `mpi/examples/benchs/recv_wait_finalize_bench.c`.

To test how much MPI priorities have a good effect on performance, you can set the environment variable STARPU_MPI_PRIORITIES to `0` to disable the use of priorities in StarPU-MPI.

43.15 MPI Cache Support

StarPU-MPI automatically optimizes duplicate data transmissions: if an MPI node `B` needs a piece of data `D` from MPI node `A` for several tasks, only one transmission of `D` will take place from `A` to `B`, and the value of `D` will be kept on `B` as long as no task modifies `D`.

If a task modifies `D`, `B` will wait for all tasks which need the previous value of `D`, before invalidating the value of `D`. As a consequence, it releases the memory occupied by `D`. Whenever a task running on `B` needs the new value of `D`, allocation will take place again to receive it.

Since tasks can be submitted dynamically, StarPU-MPI can not know whether the current value of data `D` will again be used by a newly-submitted task before being modified by another newly-submitted task, so until a task is submitted to modify the current value, it can not decide by itself whether to flush the cache or not. The application can however explicitly tell StarPU-MPI to flush the cache by calling starpu_mpi_cache_flush() or starpu_mpi_cache_flush_all_data(), for instance in case the data will not be used at all anymore (see for instance the cholesky example in `mpi/examples/matrix_decomposition`), or at least not in the close future. If a newly-submitted task actually needs the value again, another transmission of `D` will be initiated from `A` to `B`. A mere starpu_mpi_cache_flush_all_data() can for instance be added at the end of the whole algorithm, to express that no data will be reused after this (or at least that it is not interesting to keep them in cache). It may however be interesting to add fine-graph starpu_mpi_cache_flush() calls during the algorithm; the effect for the data deallocation will be the same, but it will additionally release some pressure from the StarPU-MPI cache hash table during task submission.

One can determine whether a piece of data is cached with starpu_mpi_cached_receive() and starpu_mpi_cached_send().

An example is available in the file `mpi/examples/cache/cache.c`.

Functions starpu_mpi_cached_receive_set() and starpu_mpi_cached_send_set() are automatically called by starpu_mpi_task_insert() but can also be called directly by the application. Functions starpu_mpi_cached_send_clear() and starpu_mpi_cached_receive_clear() must be called to clear data from the cache. They are also automatically called when using starpu_mpi_task_insert().

The whole caching behavior can be disabled thanks to the STARPU_MPI_CACHE environment variable. The variable STARPU_MPI_CACHE_STATS can be set to `1` to enable the runtime to display messages when data are added or removed from the cache holding the received data.

43.16 MPI Data Migration

The application can dynamically change its mind about the data distribution, to balance the load over MPI nodes, for instance. This can be done very simply by requesting an explicit move and then change the registered rank. For instance, we here switch to a new distribution function `my_distrib2`: we first register any data which wasn't registered already and will be needed, then migrate the data, and register the new location.

```
for(x = 0; x < X; x++)
{
    for (y = 0; y < Y; y++)
    {
        int mpi_rank = my_distrib2(x, y, size);
        if (!data_handles[x][y] && (mpi_rank == my_rank
              || my_rank == my_distrib(x+1, y, size) || my_rank == my_distrib(x-1, y, size)
              || my_rank == my_distrib(x, y+1, size) || my_rank == my_distrib(x, y-1, size)))
            /* Register newly-needed data */
            starpu_variable_data_register(&data_handles[x][y], -1, (uintptr_t)NULL, sizeof(unsigned));
        if (data_handles[x][y])
        {
            /* Migrate the data */
            starpu_mpi_data_migrate(MPI_COMM_WORLD, data_handles[x][y], mpi_rank);
        }
    }
}
```

The full example is available in the file `mpi/examples/stencil/stencil5.c`. From then on, further tasks submissions will use the new data distribution, which will thus change both MPI communications and task assignments.

Very importantly, since all nodes have to agree on which node owns which data to determine MPI communications and task assignments the same way, all nodes have to perform the same data migration, and at the same point among task submissions. It thus does not require a strict synchronization, just a clear separation of task submissions before and after the data redistribution.

Before data unregistration, it has to be migrated back to its original home node (the value, at least), since that is where the user-provided buffer resides. Otherwise, the unregistration will complain that it does not have the latest value on the original home node.

```
for(x = 0; x < X; x++)
{
    for (y = 0; y < Y; y++)
    {
        if (data_handles[x][y])
        {
            int mpi_rank = my_distrib(x, y, size);
            /* Get back data to original place where the user-provided buffer is.    */
            starpu_mpi_data_migrate(MPI_COMM_WORLD, data_handles[x][y], mpi_rank);
            /* And unregister it */
            starpu_data_unregister(data_handles[x][y]);
        }
    }
}
```

43.17 MPI Collective Operations

The functions are described in MPICollectiveOperations.

```
if (rank == root)
{
    /* Allocate the vector */
    vector = malloc(nblocks * sizeof(float *));
    for(x=0 ; x<nblocks ; x++)
    {
        starpu_malloc((void **)&vector[x], block_size*sizeof(float));
    }
}
/* Allocate data handles and register data to StarPU */
data_handles = malloc(nblocks*sizeof(starpu_data_handle_t *));
for(x = 0; x < nblocks ;  x++)
{
    int mpi_rank = my_distrib(x, nodes);
    if (rank == root)
    {
        starpu_vector_data_register(&data_handles[x], STARPU_MAIN_RAM, (uintptr_t)vector[x], blocks_size,
      sizeof(float));
    }
    else if ((mpi_rank == rank) || ((rank == mpi_rank+1 || rank == mpi_rank-1)))
    {
        /* I own this index, or i will need it for my computations */
        starpu_vector_data_register(&data_handles[x], -1, (uintptr_t)NULL, block_size, sizeof(float));
    }
    else
    {
        /* I know it's useless to allocate anything for this */
        data_handles[x] = NULL;
    }
    if (data_handles[x])
    {
        starpu_mpi_data_register(data_handles[x], x*nblocks+y, mpi_rank);
    }
}
/* Scatter the matrix among the nodes */
starpu_mpi_scatter_detached(data_handles, nblocks, root, MPI_COMM_WORLD, NULL, NULL, NULL, NULL);
/* Calculation */
for(x = 0; x < nblocks ;  x++)
{
    if (data_handles[x])
    {
        int owner = starpu_data_get_rank(data_handles[x]);
        if (owner == rank)
        {
            starpu_task_insert(&cl, STARPU_RW, data_handles[x], 0);
        }
    }
}
/* Gather the matrix on main node */
starpu_mpi_gather_detached(data_handles, nblocks, 0, MPI_COMM_WORLD, NULL, NULL, NULL, NULL);
```

An example is available in `mpi/tests/mpi_scatter_gather.c`.

With NewMadeleine (see Using the NewMadeleine communication library), broadcasts can automatically be detected and be optimized by using routing trees. This behavior can be controlled with the environment variable STARPU_MPI_COOP_SENDS. See the corresponding `paper` for more information.

Other collective operations would be easy to define, just ask starpu-devel for them!

43.18 Make StarPU-MPI Progression Thread Execute Tasks

The default behavior of StarPU-MPI is to spawn an MPI thread to take care only of MPI communications in an active fashion (i.e. the StarPU-MPI thread sleeps only when there are no active request submitted by the application), with the goal of being as reactive as possible to communications. Knowing that, users usually leave one free core for the MPI thread when starting a distributed execution with StarPU-MPI. However, this could result in a loss of performance for applications that does not require an extreme reactivity to MPI communications.

The starpu_mpi_init_conf() routine allows users to give the starpu_conf configuration structure of StarPU (usually given to the starpu_init() routine) to StarPU-MPI, so that StarPU-MPI reserves for its own use one of the CPU drivers of the current computing node, or one of the CPU cores, and then calls starpu_init() internally.

This allows the MPI communication thread to call a StarPU CPU driver to run tasks when there is no active requests to take care of, and thus recover the computational power of the "lost" core. Since there is a trade-off between executing tasks and polling MPI requests, which is how much the application wants to lose in reactivity to MPI communications to get back the computing power of the core dedicated to the StarPU-MPI thread, there are two environment variables to pilot the behavior of the MPI thread so that users can tune this trade-off depending on the behavior of the application.

The STARPU_MPI_DRIVER_CALL_FREQUENCY environment variable sets how many times the MPI progression thread goes through the MPI_Test() loop on each active communication request (and thus try to make communications progress by going into the MPI layer) before executing tasks. The default value for this environment variable is 0, which means that the support for interleaving task execution and communication polling is deactivated, thus returning the MPI progression thread to its original behavior.

The STARPU_MPI_DRIVER_TASK_FREQUENCY environment variable sets how many tasks are executed by the MPI communication thread before checking all active requests again. While this environment variable allows a better use of the core dedicated to StarPU-MPI for computations, it also decreases the reactivity of the MPI communication thread as much.

43.19 Debugging MPI

Communication trace will be enabled when the environment variable STARPU_MPI_COMM is set to `1`, and StarPU has been configured with the option --enable-verbose.

Statistics will be enabled for the communication cache when the environment variable STARPU_MPI_CACHE_STATS is set to `1`. It prints messages on the standard output when data are added or removed from the received communication cache.

When the environment variable STARPU_MPI_STATS is set to `1`, StarPU will display at the end of the execution for each node the volume and the bandwidth of data sent to all the other nodes. Communication statistics can also be enabled and disabled from the application by calling the functions starpu_mpi_comm_stats_enable() and starpu_mpi_comm_stats_disable(). If communication statistics have been enabled, calling the function starpu_mpi_comm_stats_retrieve() will give the amount of communications between the calling node and all the other nodes. Communication statistics will also be automatically displayed at the end of the execution, as exampli-fied below.

```
[starpu_comm_stats][3] TOTAL:   476.000000 B    0.000454 MB      0.000098 B/s     0.000000 MB/s
[starpu_comm_stats][3:0]        248.000000 B    0.000237 MB      0.000051 B/s     0.000000 MB/s
[starpu_comm_stats][3:2]        50.000000 B     0.000217 MB      0.000047 B/s     0.000000 MB/s

[starpu_comm_stats][2] TOTAL:   288.000000 B    0.000275 MB      0.000059 B/s     0.000000 MB/s
[starpu_comm_stats][2:1]        70.000000 B     0.000103 MB      0.000022 B/s     0.000000 MB/s
[starpu_comm_stats][2:3]        288.000000 B    0.000172 MB      0.000037 B/s     0.000000 MB/s

[starpu_comm_stats][1] TOTAL:   188.000000 B    0.000179 MB      0.000038 B/s     0.000000 MB/s
[starpu_comm_stats][1:0]        80.000000 B     0.000114 MB      0.000025 B/s     0.000000 MB/s
[starpu_comm_stats][1:2]        188.000000 B    0.000065 MB      0.000014 B/s     0.000000 MB/s

[starpu_comm_stats][0] TOTAL:   376.000000 B    0.000359 MB      0.000077 B/s     0.000000 MB/s
[starpu_comm_stats][0:1]        376.000000 B    0.000141 MB      0.000030 B/s     0.000000 MB/s
[starpu_comm_stats][0:3]        10.000000 B     0.000217 MB      0.000047 B/s     0.000000 MB/s
```

These statistics can be plotted as heatmaps using the StarPU tool `starpu_mpi_comm_matrix.py`, this will produce 2 PDF files, one plot for the bandwidth, and one plot for the data volume.

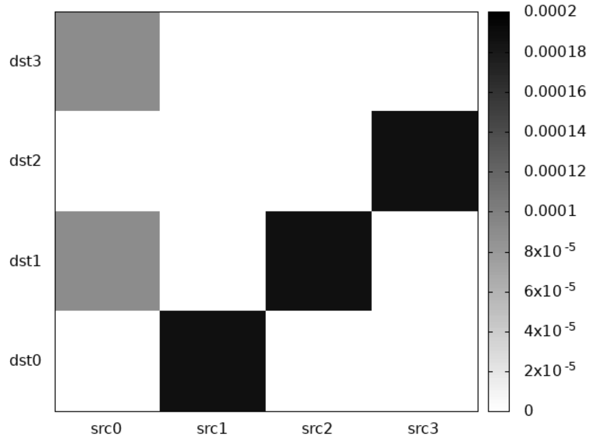


Figure 43.1 Bandwidth Heatmap

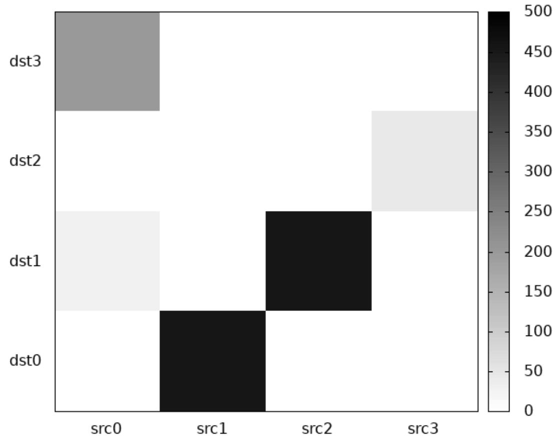


Figure 43.2 Data Volume Heatmap

43.20 More MPI examples

MPI examples are available in the StarPU source code in mpi/examples:

- `comm` shows how to use communicators with StarPU-MPI
- `complex` is a simple example using a user-define data interface over MPI (complex numbers),
- `stencil5` is a simple stencil example using starpu_mpi_task_insert(),
- `matrix_decomposition` is a cholesky decomposition example using starpu_mpi_task_insert(). The non-distributed version can check for <algorithm correctness in 1-node configuration, the distributed version uses exactly the same source code, to be used over MPI,
- `mpi_lu` is an LU decomposition example, provided in three versions: `plu_example` uses explicit MPI data transfers, `plu_implicit_example` uses implicit MPI data transfers, `plu_outofcore_example` uses implicit MPI data transfers and supports data matrices which do not fit in memory (out-of-core).

43.21 Using the NewMadeleine communication library

NewMadeleine (see `https://pm2.gitlabpages.inria.fr/newmadeleine/`, part of the PM2 project) is an optimizing communication library for high-performance networks. NewMadeleine provides its own interface, but also an MPI interface (called MadMPI). Thus, there are two possibilities to use NewMadeleine with StarPU:

- using the NewMadeleine's native interface. StarPU supports this interface from its release 1.3.0, by enabling the `configure` option --enable-nmad. In this case, StarPU relies directly on NewMadeleine to make communications progress and NewMadeleine has to be built with the profile `pukabi+madmpi.conf`.
- using the NewMadeleine's MPI interface (MadMPI). StarPU will use the standard MPI API and NewMadeleine will handle the calls to the MPI API. In this case, StarPU makes communications progress and thus communication progress has to be disabled in NewMadeleine by compiling it with the profile `pukabi+madmpi-mini.conf`.

To build NewMadeleine, download the latest version from the website (or, better, use the Git version to use the most recent version), then:

```
cd pm2/scripts
./pm2-build-packages ./<the profile you chose> --prefix=<installation prefix>
```

With Guix, the NewMadeleine's native interface can be used by setting the parameter `--with-input=openmpi=nmad` and MadMPI can be used with `--with-input=openmpi=nmad-mini`.

Whatever implementation (NewMadeleine or MadMPI) is used by StarPU, the public MPI interface of StarPU (described in MPI Support) is the same.

43.22 MPI Master Slave Support

StarPU provides another way to execute applications across many nodes. The Master Slave support permits to use remote cores without thinking about data distribution. This support can be activated with the `configure` option --enable-mpi-master-slave. However, you should not activate both MPI support and MPI Master-Slave support.

The existing kernels for CPU devices can be used as such. They only have to be exposed through the name of the function in the starpu_codelet::cpu_funcs_name field. Functions have to be globally-visible (i.e. not static) for StarPU to be able to look them up, and `-rdynamic` must be passed to gcc (or `-export-dynamic` to ld) so that symbols of the main program are visible.

By default, one core is dedicated on the master node to manage the entire set of slaves. If the implementation of MPI you are using has a good multiple threads support, you can set the STARPU_MPI_MS_MULTIPLE_THREAD environment variable to 1 to dedicate one core per slave.

Choosing the number of cores on each slave device is done by setting the environment variable STARPU_NMPIMSTHREADS=<numb with `<number>` being the requested number of cores. By default, all the slave's cores are used.

Setting the number of slaves nodes is done by changing the `-np` parameter when executing the application with mpirun or mpiexec.

The master node is by default the node with the MPI rank equal to 0. To select another node, use the environment variable STARPU_MPI_MASTER_NODE=<number> with `<number>` being the requested MPI rank node.

A simple example `tests/main/insert_task.c` can be used to test the MPI master slave support.

43.23 MPI Checkpoint Support

StarPU provides an experimental checkpoint mechanism. It is for now only a proof of concept to see what the checkpointing cost is, since the restart part has not been integrated yet.
To enable checkpointing, you should use the `configure` option --enable-mpi-ft. The application in the directory `mpi/examples/matrix_decomposition` shows how to enable checkpoints. The API documentation is available in MPI Fault Tolerance Support
Statistics can also be enabled with the `configure` option --enable-mpi-ft-stats.

Chapter 44

TCP/IP Support

44.1 TCP/IP Master Slave Support

StarPU provides a transparent way to execute applications across many nodes. The Master Slave support permits to use remote cores without thinking about data distribution. This support can be activated with the `configure` option --enable-tcpip-master-slave.

The existing kernels for CPU devices can be used as such. They only have to be exposed through the name of the function in the starpu_codelet::cpu_funcs_name field. Functions have to be globally-visible (i.e. not static) for StarPU to be able to look them up, and `-rdynamic` must be passed to gcc (or `-export-dynamic` to ld) so that symbols of the main program are visible.

By default, one core is dedicated on the master node to manage the entire set of slaves.

Choosing the number of cores on each slave device is done by setting the environment variable STARPU_NTCPIPMSTHREADS=<nu
with `<number>` being the requested number of cores. By default, all the slave's cores are used.

The master should be given the number of slaves that are expected to be run with the STARPU_TCPIP_MS_SLAVES environment variable.

The slaves should then be started, and their number also should be given with the STARPU_TCPIP_MS_SLAVES environment variable. They should additionally be given the IP address of the master with the STARPU_TCPIP_MS_MASTER environment variable.

For simple local checks, one can use the `starpu_tcpipexec` tool, which just starts the application several times. Setting the number of slaves nodes is done by changing the `-np` parameter.

Chapter 45

Transactions

45.1 General Ideas

StarPU's transactions enable the cancellation of a sequence of already submitted tasks based on a just-in-time decision. The purpose of this mechanism is typically for iterative applications to submit tasks for the next iteration ahead of time while leaving some iteration loop criterion (e.g. convergence) to be evaluated just before the first task of the next iteration is about to be scheduled. Such a sequence of collectively cancelable tasks is called a transaction *epoch*.

45.2 Usage

Some examples illustrating the usage of StarPU's transactions are available in the directory `examples/transactions`.

45.2.1 Epoch Cancellation

If the start criterion of an epoch evaluates to `False`, all the tasks for that next epoch are canceled. Thus, StarPU's transactions let applications avoid the use of synchronization barriers commonly found between the task submission sequences of subsequent iterations, and avoid breaking the flow of dependencies in the process. Moreover, while the kernel functions of canceled transaction tasks are not executed, their dependencies are still honored in the proper order.

45.2.2 Transactions Enabled Codelets

Codelets for tasks being part of a transaction should set their `nbuffers` field to STARPU_VARIABLE_NBUFFERS.

45.2.3 Transaction Creation

A `struct starpu_transaction` opaque object is created using the starpu_transaction_open() function, specifying a transaction start criterion callback and some user argument to be passed to that callback upon the first call. The start criterion callback should return `True` (e.g. `!0`) if the next transaction epoch should proceed, or `False` (e.g. `0`) if the tasks belonging to that next epoch should be canceled. `starpu_transaction_open()` submits an internal task to mark the beginning of the transaction. If submitting that internal task fails with ENODEV, `starpu_transaction_open()` will return `NULL`.

45.2.4 Transaction Tasks

Tasks governed by the same transaction object should be passed that transaction object either through the .transaction field of starpu_task structures, using the STARPU_TRANSACTION argument of starpu_task_insert().

45.2.5 Epoch Transition

The transition from one transaction epoch to the next is expressed using the starpu_transaction_next_epoch function to which the `starpu_transaction` object and a user argument are passed. Upon a call to that function, the start criterion callback is evaluated on users argument to decide whether the next epoch should proceed or be canceled.

45.2.6 Transaction Closing

The last epoch should be ended through a call to starpu_transaction_close().

45.3 Known limitations

Support for transactions is experimental.
StarPU's transactions are currently not compatible with StarPU-MPI distributed sessions.

Chapter 46

Fault Tolerance

46.1 Introduction

Due to e.g. hardware error, some tasks may fail, or even complete nodes may fail. For now, StarPU provides some support for failure of tasks.

46.2 Retrying tasks

In case a task implementation notices that it fail to compute properly, it can call starpu_task_failed() to notify StarPU of the failure.

`tests/fault-tolerance/retry.c` is an example of coping with such failure: the principle is that when submitting the task, one sets its prologue callback to starpu_task_ft_prologue(). That prologue will turn the task into a meta task, which will manage the repeated submission of try-tasks to perform the computation until one of the computations succeeds. One can create a try-task for the meta task by using starpu_task_ft_create_retry().

By default, try-tasks will be just retried until one of them succeeds (i.e. the task implementation does not call starpu_task_failed()). One can change the behavior by passing a `check_failsafe` function as prologue parameter, which will be called at the end of the try-task attempt. It can look at `starpu_task_get_current()->failed` to determine whether the try-task succeeded, in which case it can call starpu_task_ft_success() on the meta-task to notify success, or if it failed, in which case it can call starpu_task_failsafe_create_retry() to create another try-task, and submit it with starpu_task_submit_nodeps().

This can however only work if the task input is not modified, and is thus not supported for tasks with data access mode STARPU_RW.

Chapter 47

FFT Support

StarPU provides `libstarpufft`, a library whose design is very similar to both `fftw` and `cufft`, the difference being that it takes benefit from both CPUs and GPUs. It should however be noted that GPUs do not have the same precision as CPUs, so the results may be different by a negligible amount.
Different precisions are available, namely `float`, `double` and `long double` precisions, with the following `fftw` naming conventions:

- double precision structures and functions are named e.g. starpufft_execute()
- float precision structures and functions are named e.g. starpufftf_execute()
- long double precision structures and functions are named e.g. starpufftl_execute()

The documentation below is given with names for double precision, replace `starpufft_` with `starpufftf_` or `starpufftl_` as appropriate.
Only complex numbers are supported at the moment.
The application has to call starpu_init() before calling `starpufft` functions.
Either main memory pointers or data handles can be provided.

- To provide main memory pointers, use starpufft_start() or starpufft_execute(). Only one FFT can be performed at a time, because StarPU will have to register the data on the fly. In the starpufft_start() case, starpufft_cleanup() needs to be called to unregister the data.
- To provide data handles (which is preferable), use starpufft_start_handle() (preferred) or starpufft_execute_handle(). Several FFTs tasks can be submitted for a given plan, which permits e.g. to start a series of FFT with just one plan. starpufft_start_handle() is preferable since it does not wait for the task completion, and thus permits to enqueue a series of tasks.

All functions are defined in FFT Support.
Some examples illustrating the usage of FFT API are available in the directory `starpufft/tests`.

47.1 Compilation

The flags required to compile or link against the FFT library are accessible with the following commands:

```
$ pkg-config --cflags starpufft-1.4  # options for the compiler
$ pkg-config --libs starpufft-1.4    # options for the linker
```

Also pass the option `-static` if the application is to be linked statically.

Chapter 48

SOCL OpenCL Extensions

SOCL is an OpenCL implementation based on StarPU. It gives unified access to every available OpenCL device : applications can now share entities such as Events, Contexts or Command Queues between several OpenCL implementations.

In addition, command queues that are created without specifying a device provide automatic scheduling of the submitted commands on OpenCL devices contained in the context to which the command queue is attached.

Setting the `CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE` flag on a command queue also allows StarPU to reorder kernels queued on the queue, otherwise they would be serialized, and several command queues would be necessary to see kernels dispatched to the various OpenCL devices.

Note: this is still an area under development and subject to change.

When compiling StarPU, SOCL will be enabled if a valid OpenCL implementation is found on your system. To be able to run the SOCL test suite, the environment variable SOCL_OCL_LIB_OPENCL needs to be defined to the location of the file `libOpenCL.so` of the OCL ICD implementation. You should for example add the following line in your file `.bashrc`

```
export SOCL_OCL_LIB_OPENCL=/usr/lib/x86_64-linux-gnu/libOpenCL.so
```

You can then run the test suite in the directory `socl/examples`.

```
$ make check
...
PASS: basic/basic
PASS: testmap/testmap
PASS: clinfo/clinfo
PASS: matmul/matmul
PASS: mansched/mansched
==================
All 5 tests passed
==================
```

The environment variable OCL_ICD_VENDORS has to point to the directory where the socl.icd ICD file is installed. When compiling StarPU, the files are in the directory `socl/vendors`. With an installed version of StarPU, the files are installed in the directory `$prefix/share/starpu/opencl/vendors`.

To run the tests by hand, you have to call, for example,

```
$ LD_PRELOAD=$SOCL_OCL_LIB_OPENCL OCL_ICD_VENDORS=socl/vendors/ socl/examples/clinfo/clinfo
Number of platforms:    2
  Platform Profile:     FULL_PROFILE
  Platform Version:     OpenCL 1.1 CUDA 4.2.1
  Platform Name:        NVIDIA CUDA
  Platform Vendor:      NVIDIA Corporation
  Platform Extensions:  cl_khr_byte_addressable_store cl_khr_icd cl_khr_gl_sharing cl_nv_compiler_options cl_r

  Platform Profile:     FULL_PROFILE
  Platform Version:     OpenCL 1.0 SOCL Edition (0.1.0)
  Platform Name:        SOCL Platform
  Platform Vendor:      Inria
  Platform Extensions:  cl_khr_icd
....
$
```

To enable the use of CPU cores via OpenCL, one can set the STARPU_OPENCL_ON_CPUS environment variable to 1 and STARPU_NCPUS to 0 (to avoid using CPUs both via the OpenCL driver and the normal CPU driver).

Chapter 49

Hierarchical DAGS

The STF model has the intrinsic limitation of supporting static task graphs only, which leads to potential submission overhead and to a static task graph which is not necessarily adapted for execution on heterogeneous systems.
To address these problems, we have extended the STF model to enable tasks subgraphs at runtime. We refer to these tasks as *hierarchical tasks*. This approach allows for a more dynamic task graph. This allows to dynamically adapt the granularity to meet the optimal size of the targeted computing resource.
Hierarchical tasks are tasks that can transform themselves into a new task-graph dynamically at runtime. Programmers submit a coarse version of the DAG, called the bubbles graph, which represents the general shape of the application tasks graph. The execution of this bubble graph will generate and submit the computing tasks of the application. It is up to application programmers to decide how to build the bubble graph (i.e. how to structure the computation tasks graph to create some groups of tasks). Dependencies between bubbles are automatically deduced from dependencies between their computing tasks.

49.1 An Example

In order to understand the hierarchical tasks model, an example of "bubblification" is showed here. We start from a simple example, multiplying the elements of a vector.

49.1.1 Initial Version

A computation is done several times on a vector split in smaller vectors. For each step and each sub-vector, a task is generated to perform the computation.

```
void func_cpu(void *descr[], void *_args)
{
        (void) _args;
        int x;
        int nx = STARPU_VECTOR_GET_NX(descr[0]);
        TYPE *v = (TYPE *)STARPU_VECTOR_GET_PTR(descr[0]);
        for(x=0 ; x<nx ; x++)
                v[x] += 1;
}
struct starpu_codelet vector_cl =
{
        .cpu_funcs = {func_cpu},
        .nbuffers = 1,
        .modes = {STARPU_RW}
};
int vector_no_bubble()
{
        TYPE *vector;
        starpu_data_handle_t vhandle;
        /* ...  */
        starpu_vector_data_register(&vhandle, 0, (uintptr_t)vector, X, sizeof(vector[0]));
        starpu_data_map_filters(vhandle, 1, &f);
        for(loop=0 ; loop<NITER; loop++)
                for (x = 0; x < SLICES; x++)
                {
                        starpu_task_insert(&vector_cl,
                                           STARPU_RW, starpu_data_get_sub_data(vhandle, 1, x),
                                           0);
                }
        starpu_data_unpartition(vhandle, STARPU_MAIN_RAM);
        starpu_data_unregister(vhandle);
        /* ...  */
}
```

49.1.2 Bubble Version

The bubble version of the code replaces the inner loop that realizes the tasks insertion by a call to a bubble creation. At its execution, the bubble will insert the computing tasks. The bubble graph is built accordingly to the dependencies of the subdata.

```
void no_func(void *buffers[], void *arg)
{
        assert(0);
        return;
}
int is_bubble(struct starpu_task *t, void *arg)
{
        (void)arg;
        (void)t;
        return 1;
}
void bubble_gen_dag(struct starpu_task *t, void *arg)
{
        int i;
        starpu_data_handle_t *subdata = (starpu_data_handle_t *)arg;
        for(i=0 ; i<SLICES ; i++)
        {
                starpu_task_insert(&vector_cl,
                                STARPU_RW, subdata[i],
                                0);
                STARPU_CHECK_RETURN_VALUE(ret, "starpu_task_insert");
        }
}
struct starpu_codelet bubble_codelet =
{
        .cpu_funcs = {no_func},
        .bubble_func = is_bubble,
        .bubble_gen_dag_func = bubble_gen_dag,
        .nbuffers = 1
};
int vector_bubble()
{
        TYPE *vector;
        starpu_data_handle_t vhandle;
        starpu_data_handle_t sub_handles[SLICES];
        /* ...  */
        starpu_vector_data_register(&vhandle, 0, (uintptr_t)vector, X, sizeof(vector[0]));
        starpu_data_partition_plan(vhandle, &f, sub_handles);
        for(loop=0 ; loop<NITER; loop++)
        {
                starpu_task_insert(&bubble_codelet,
                                STARPU_RW, vhandle,
                                STARPU_NAME, "B1",
                                STARPU_BUBBLE_GEN_DAG_FUNC_ARG, sub_handles,
                                0);
        }
        starpu_data_partition_clean(vhandle, SLICES, sub_handles);
        starpu_data_unregister(vhandle);
        /* ...  */
}
```

The full example is available in the file `bubble/tests/vector/vector.c`.

To define a hierarchical task, one needs to define the fields starpu_codelet::bubble_func and starpu_codelet::bubble_gen_dag_func.

The field starpu_codelet::bubble_func is a pointer function which will be executed by StarPU to decide at runtime if the task must be transformed into a bubble. If the function returns a non-zero value, the function starpu_codelet::bubble_gen_dag_func will be executed to create the new graph of tasks.

The pointer functions can also be defined when calling starpu_task_insert() by using the arguments STARPU_BUBBLE_FUNC and STARPU_BUBBLE_GEN_DAG_FUNC. Both these functions can be passed parameters through the arguments STARPU_BUBBLE_FUNC_ARG and STARPU_BUBBLE_GEN_DAG_FUNC_ARG

When executed, the function starpu_codelet::bubble_func will be given as parameter the task being checked, and the value specified with STARPU_BUBBLE_FUNC_ARG.

When executed, the function starpu_codelet::bubble_gen_dag_func will be given as parameter the task being turned into a hierarchical task and the value specified with STARPU_BUBBLE_GEN_DAG_FUNC_ARG.

An example involving these functions is in `bubble/tests/basic/brec.c`. And more examples are available in `bubble/tests/basic/*.c`.

Chapter 50

Parallel Workers

50.1 General Ideas

Parallel workers are a concept introduced in this `paper` where they are called clusters.

The granularity problem is tackled by using resource aggregation: instead of dynamically splitting tasks, resources are aggregated to process coarse grain tasks in a parallel fashion. This is built on top of scheduling contexts to be able to handle any type of parallel tasks.

This comes from a basic idea, making use of two levels of parallelism in a DAG. We keep the DAG parallelism, but consider on top of it that a task can contain internal parallelism. A good example is if each task in the DAG is OpenMP enabled.

The particularity of such tasks is that we will combine the power of two runtime systems: StarPU will manage the DAG parallelism and another runtime (e.g. OpenMP) will manage the internal parallelism. The challenge is in creating an interface between the two runtime systems so that StarPU can regroup cores inside a machine (creating what we call a **parallel worker**) on top of which the parallel tasks (e.g. OpenMP tasks) will be run in a contained fashion.

The aim of the parallel worker API is to facilitate this process automatically. For this purpose, we depend on the `hwloc` tool to detect the machine configuration and then partition it into usable parallel workers.

An example of code running on parallel workers is available in `examples/sched_ctx/parallel_workers.c`.

Let's first look at how to create a parallel worker.

To enable parallel workers in StarPU, one needs to set the configure option --enable-parallel-worker.

50.2 Workers Creating Parallel Workers

Partitioning a machine into parallel workers with the parallel worker API is fairly straightforward. The simplest way is to state under which machine topology level we wish to regroup all resources. This level is a `hwloc` object, of the type `hwloc_obj_type_t`. More information can be found in the `hwloc documentation`.

Once a parallel worker is created, the full machine is represented with an opaque structure starpu_parallel_worker_config. This can be printed to show the current machine state.

```
struct starpu_parallel_worker_config *parallel_workers;
parallel_workers = starpu_parallel_worker_init(HWLOC_OBJ_SOCKET, 0);
starpu_parallel_worker_print(parallel_workers);
/* submit some tasks with OpenMP computations */
starpu_parallel_worker_shutdown(parallel_workers);
/* we are back to the default StarPU state */
```

The following graphic is an example of what a particular machine can look like once parallel workers are created. The main difference is that we have less worker queues and tasks which will be executed on several resources at once. The execution of these tasks will be left to the internal runtime system, represented with a dashed box around the resources.

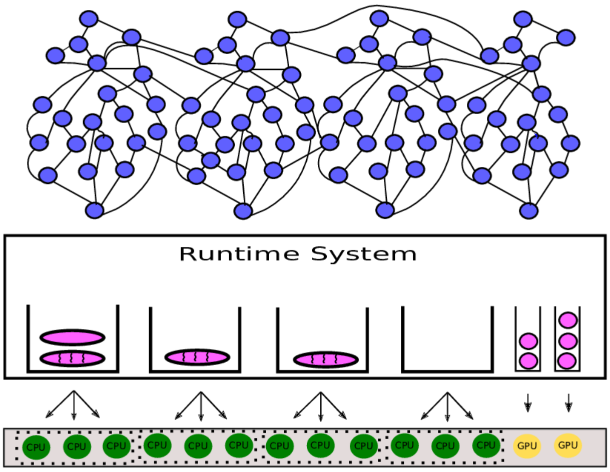


Figure 50.1 StarPU using parallel tasks

Creating parallel workers as shown in the example above will create workers able to execute OpenMP code by default. The parallel worker creation function starpu_parallel_worker_init() takes optional parameters after the `hwloc` object (always terminated by the value `0`) which allow parametrizing the parallel workers creation. These parameters can help to create parallel workers of a type different from OpenMP, or create a more precise partition of the machine.
This is explained in Section Creating Custom Parallel Workers.
Before starpu_shutdown(), we call starpu_parallel_worker_shutdown() to delete the parallel worker configuration.

50.3 Example Of Constraining OpenMP

Parallel workers require being able to constrain the runtime managing the internal task parallelism (internal runtime) to the resources set by StarPU. The purpose of this is to express how StarPU must communicate with the internal runtime to achieve the required cooperation. In the case of OpenMP, StarPU will provide an awake thread from the parallel worker to execute this liaison. It will then provide on demand the process ids of the other resources supposed to be in the region. Finally, thanks to an OpenMP region, we can create the required number of threads and bind each of them on the correct region. These will then be reused each time we encounter a `#pragma omp parallel` in the following computations of our program.
The following graphic is an example of what an OpenMP-type parallel worker looks like and how it is represented in StarPU. We can see that one StarPU (black) thread is awake, and we need to create on the other resources the OpenMP threads (in pink).

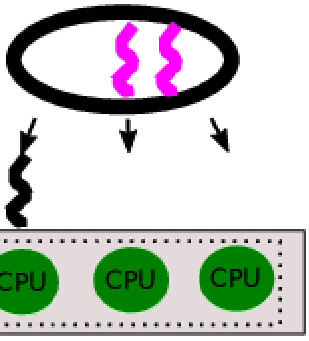


Figure 50.2 StarPU with an OpenMP parallel worker

Finally, the following code shows how to force OpenMP to cooperate with StarPU and create the aforementioned OpenMP threads constrained in the parallel worker's resources set:

```
void starpu_parallel_worker_openmp_prologue(void * sched_ctx_id)
{
  int sched_ctx = *(int*)sched_ctx_id;
  int *cpuids = NULL;
  int ncpuids = 0;
  int workerid = starpu_worker_get_id();
  //we can target only CPU workers
  if (starpu_worker_get_type(workerid) == STARPU_CPU_WORKER)
  {
    //grab all the ids inside the parallel worker
    starpu_sched_ctx_get_available_cpuids(sched_ctx, &cpuids, &ncpuids);
    //set the number of threads
    omp_set_num_threads(ncpuids);
#pragma omp parallel
    {
      //bind each threads to its respective resource
      starpu_sched_ctx_bind_current_thread_to_cpuid(cpuids[omp_get_thread_num()]);
    }
    free(cpuids);
  }
  return;
}
```

This function is the default function used when calling starpu_parallel_worker_init() without extra parameter.
Parallel workers are based on several tools and models already available within StarPU contexts, and merely extend contexts. More on contexts can be read in Section Scheduling Contexts.
A similar example is available in the file `examples/sched_ctx/parallel_code.c`.

50.4 Creating Custom Parallel Workers

Parallel workers can be created either with the predefined types provided within StarPU, or with user-defined functions to bind another runtime inside StarPU.
The predefined parallel worker types provided by StarPU are STARPU_PARALLEL_WORKER_OPENMP, STARPU_PARALLEL_WORKER_INTEL_OPENMP_MKL and STARPU_PARALLEL_WORKER_GNU_OPENMP_MKL.
If StarPU is compiled with the `MKL` library, STARPU_PARALLEL_WORKER_GNU_OPENMP_MKL uses MKL functions to set the number of threads, which is more reliable when using an OpenMP implementation different from the Intel one. Otherwise, it will behave as STARPU_PARALLEL_WORKER_INTEL_OPENMP_MKL.
The parallel worker type is set when calling the function starpu_parallel_worker_init() with the parameter STARPU_PARALLEL_WORKER_TYPE as in the example below, which is creating a `MKL` parallel worker.

```
struct starpu_parallel_worker_config *parallel_workers;
parallel_workers = starpu_parallel_worker_init(HWLOC_OBJ_SOCKET,
                                 STARPU_PARALLEL_WORKER_TYPE, STARPU_PARALLEL_WORKER_GNU_OPENMP_MKL,
                                 0);
```

Using the default type STARPU_PARALLEL_WORKER_OPENMP is similar to calling starpu_parallel_worker_init() without any extra parameter.
An example is available in `examples/parallel_workers/parallel_workers.c`.
Users can also define their own function.

```
void foo_func(void* foo_arg);
int foo_arg = 0;
struct starpu_parallel_worker_config *parallel_workers;
parallel_workers = starpu_parallel_worker_init(HWLOC_OBJ_SOCKET,
                                  STARPU_PARALLEL_WORKER_CREATE_FUNC, &foo_func,
                                  STARPU_PARALLEL_WORKER_CREATE_FUNC_ARG, &foo_arg,
                                  0);
```

An example is available in `examples/parallel_workers/parallel_workers_func.c`.
Parameters that can be given to starpu_parallel_worker_init() are STARPU_PARALLEL_WORKER_MIN_NB, STARPU_PARALLEL_WORKER_MAX_NB, STARPU_PARALLEL_WORKER_NB, STARPU_PARALLEL_WORKER_POLICY_NAME STARPU_PARALLEL_WORKER_POLICY_STRUCT, STARPU_PARALLEL_WORKER_KEEP_HOMOGENEOUS, STARPU_PARALLEL_WORKER_PREFERE_MIN, STARPU_PARALLEL_WORKER_CREATE_FUNC, STARPU_PARALLEL_WORK STARPU_PARALLEL_WORKER_TYPE, STARPU_PARALLEL_WORKER_AWAKE_WORKERS, STARPU_PARALLEL_WORKER_P STARPU_PARALLEL_WORKER_NEW and STARPU_PARALLEL_WORKER_NCORES.

50.5 Parallel Workers With Scheduling

As previously mentioned, the parallel worker API is implemented on top of Scheduling Contexts. Its main addition is to ease the creation of a machine CPU partition with no overlapping by using `hwloc`, whereas scheduling contexts can use any number of any type of resources.
It is therefore possible, but not recommended, to create parallel workers using the scheduling contexts API. This can be useful mostly in the most complex machine configurations, where users have to dimension precisely parallel workers by hand using their own algorithm.

```
/* the list of resources the context will manage */
int workerids[3] = {1, 3, 10};
/* indicate the list of workers assigned to it, the number of workers,
the name of the context and the scheduling policy to be used within
the context */
int id_ctx = starpu_sched_ctx_create(workerids, 3, "my_ctx", 0);
/* let StarPU know that the following tasks will be submitted to this context */
starpu_sched_ctx_set_task_context(id);
task->prologue_callback_pop_func=&runtime_interface_function_here;
/* submit the task to StarPU */
starpu_task_submit(task);
```

As this example illustrates, creating a context without scheduling policy will create a parallel worker. The interface function between StarPU and the other runtime must be specified through the field starpu_task::prologue_callback_pop_func. Such a function can be similar to the OpenMP thread team creation one (see above). An example is available in `examples/sched_ctx/parallel_tasks_reuse_handle.c`.

Note that the OpenMP mode is the default mode both for parallel workers and contexts. The result of a parallel worker creation is a woken-up master worker and sleeping "slaves" which allow the master to run tasks on their resources.

To create a parallel worker with woken-up workers, the flag STARPU_SCHED_CTX_AWAKE_WORKERS must be set when using the scheduling context API function starpu_sched_ctx_create(), or the flag STARPU_PARALLEL_WORKER_AWAKE_WORKERS must be set when using the parallel worker API function starpu_parallel worker_init().

Chapter 51

Interoperability Support

In situations where multiple parallel software elements have to coexist within the same application, uncoordinated accesses to computing units may lead such parallel software elements to collide and interfere. The purpose of the Interoperability routines of StarPU, implemented along the definition of the Resource Management APIs of Project H2020 INTERTWinE, is to enable StarPU to coexist with other parallel software elements without resulting in computing core oversubscription or undersubscription. These routines allow the programmer to dynamically control the computing resources allocated to StarPU, to add or remove processor cores and/or accelerator devices from the pool of resources used by StarPU's workers to execute tasks. They also allow multiple libraries and applicative codes using StarPU simultaneously to select distinct sets of resources independently. Internally, the Interoperability Support is built on top of Scheduling Contexts (see Scheduling Contexts).

51.1 StarPU Resource Management

The `starpurm` module is a library built on top of the `starpu` library. It exposes a series of routines prefixed with `starpurm_` defining the resource management API.
All functions are defined in Interoperability Support.

51.1.1 Linking a program with the starpurm module

The `starpurm` module must be linked explicitly with the applicative executable using it. Example Makefiles in the `starpurm/dev/` subdirectories show how to do so. If the `pkg-config` command is available and the `PKG_CONFIG_PATH` environment variable is properly positioned, the proper settings may be obtained with the following `Makefile` snippet:

```
CFLAGS += $(shell pkg-config --cflags starpurm-1.4)
LDFLAGS+= $(shell pkg-config --libs-only-L starpurm-1.4)
LDLIBS += $(shell pkg-config --libs-only-l starpurm-1.4)
```

51.1.2 Initialization and Shutdown

The `starpurm` module is initialized with a call to starpurm_initialize() and must be finalized with a call to starpurm_shutdown(). The basic example is available in `starpurm/tests/01_init_exit.c`. The `starpurm` module supports CPU cores as well as devices. An integer ID is assigned to each supported device type. The ID assigned to a given device type can be queried with the starpurm_get_device_type_id() routine, which currently expects one of the following strings as argument and returns the corresponding ID:

- `"cpu"`
- `"opencl"`
- `"cuda"`

The `cpu` pseudo device type is defined for convenience and designates CPU cores. The number of units of each type available for computation can be obtained with a call to starpurm_get_nb_devices_by_type().
Each CPU core unit available for computation is designated by its rank among the StarPU CPU worker threads and by its own CPUSET bit. Each non-CPU device unit can be designated both by its rank number in the type, and by the CPUSET bit corresponding to its StarPU device worker thread. The CPUSET of a computing unit or

its associated worker can be obtained from its type ID and rank with starpurm_get_device_worker_cpuset(), which returns the corresponding HWLOC CPUSET.
An example is available in `starpurm/tests/02_list_units.c`.

51.1.3 Default Context

The `starpurm` module assumes a default, global context, manipulated through a series of routines allowing to assign and withdraw computing units from the main StarPU context. Assigning CPU cores can be done with starpurm_assign_cpu_to_starpu() and starpurm_assign_cpu_mask_to_starpu(), and assigning device units can be done with starpurm_assign_device_to_starpu() and starpurm_assign_device_mask_to_starpu(). Conversely, withdrawing CPU cores can be done with starpurm_withdraw_cpu_from_starpu() and starpurm_withdraw_cpu_mask_from_starpu(), and withdrawing device units can be done with starpurm_withdraw_device_from_starpu() and starpurm_withdraw_device_mask_from_
These routine should typically be used to control resource usage for the main applicative code. An example is available in `starpurm/examples/block_test/block_test.c`.

51.1.4 Temporary Contexts

Besides the default, global context, `starpurm` can create temporary contexts and launch the computation of kernels confined to these temporary contexts. The routine starpurm_spawn_kernel_on_cpus() can be used to do so: it allocates a temporary context and spawns a kernel within this context. The temporary context is subsequently freed upon completion of the kernel. The temporary context is set as the default context for the kernel throughout its lifespan. This routine should typically be used to control resource usage for a parallel kernel, handled by an external library built on StarPU. Internally, it relies on the use of starpu_sched_ctx_set_context() to set the temporary context as the default context for the parallel kernel, and then restore the main context upon completion. Note: the maximum number of temporary contexts allocated concurrently at any time should not exceed STARPU_NMAX_SCHED_CTXS-2, otherwise, the call to starpurm_spawn_kernel_on_cpus() may block until a temporary context becomes available. The routine starpurm_spawn_kernel_on_cpus() returns upon the completion of the parallel kernel. An example is available in `starpurm/examples/spawn.c`. An asynchronous variant is available with the routine starpurm_spawn_kernel_on_cpus_callback(). This variant returns immediately, however it accepts a callback function, which is subsequently called to notify the calling code about the completion of the parallel kernel. An example is available in `starpurm/examples/async_spawn.c`.

Chapter 52

SimGrid Support

StarPU can use SimGrid in order to simulate execution on an arbitrary platform.

The principle is to first run the application natively on the platform that one wants to laterlater simulate, and let StarPU record performance models. One then recompiles StarPU and the application in simgrid mode, where everything is executed the same, except the execution of the codelet function, and the data transfers, which are replaced by virtual sleeps based on the performance models. This thus allows to use the performance model for tasks and data transfers, while executing natively all the rest (the task scheduler and the application, notably).

This was tested with SimGrid from 3.11 to 3.16, and 3.18 to 3.35. SimGrid version 3.25 needs to be configured with -Denable_msg=ON . Other versions may have compatibility issues. 3.17 notably does not build at all. MPI simulation does not work with version 3.22.

If you have installed SimGrid by hand, make sure to set `PKG_CONFIG_PATH` to the path where `simgrid.pc` was installed:

```
$ export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/where/simgrid/installed/lib/ppkgconfig/simgrid.pc
```

52.1 Preparing Your Application For Simulation

There are a few technical details which need to be handled for an application to be simulated through SimGrid.

If the application uses `gettimeofday()` to make its performance measurements, the real time will be used, which will be bogus. To get the simulated time, it has to use starpu_timing_now() which returns the virtual timestamp in us. A basic example is available in `tests/main/empty_task.c`.

For some technical reason, the application's .c file which contains `main()` has to be recompiled with `starpu_simgrid_wrap.h`, which in the SimGrid case will `# define main()` into `starpu_main()`, and it is `libstarpu` which will provide the real `main()` and will call the application's `main()`. Including `starpu.h` will already include `starpu_simgrid_wrap.h`, so usually you would not need to include `starpu_simgrid_wrap.h` explicitly, but if for some reason including the whole `starpu.h` header is not possible, you can include `starpu_simgrid_wrap.h` explicitly.

To be able to test with crazy data sizes, one may want to only allocate application data if the macro `STARPU_↩ SIMGRID` is not defined. Passing a `NULL` pointer to `starpu_data_register` functions is fine, data will never be read/written to by StarPU in SimGrid mode anyway.

To be able to run the application with e.g. CUDA simulation on a system which does not have CUDA installed, one can fill the starpu_codelet::cuda_funcs with (void∗)1, to express that there is a CUDA implementation, even if one does not actually provide it. StarPU will not actually run it in SimGrid mode anyway by default (unless the STARPU_CODELET_SIMGRID_EXECUTE or STARPU_CODELET_SIMGRID_EXECUTE_AND_INJECT flags are set in the codelet)

```
static struct starpu_codelet cl_potrf =
{
        .cpu_funcs = {chol_cpu_codelet_update_potrf},
        .cpu_funcs_name = {"chol_cpu_codelet_update_potrf"},
#ifdef STARPU_USE_CUDA
        .cuda_funcs = {chol_cublas_codelet_update_potrf},
#elif defined(STARPU_SIMGRID)
        .cuda_funcs = {(void*)1},
#endif
        .nbuffers = 1,
        .modes = {STARPU_RW},
        .model = &chol_model_potrf
};
```

The full example is available in `examples/cholesky/cholesky_kernels.c`.

52.2 Calibration

The idea is to first compile StarPU normally, and run the application, to automatically benchmark the bus and the codelets.

```
$ ./configure && make
$ STARPU_SCHED=dmda ./examples/matvecmult/matvecmult
[starpu][_starpu_load_history_based_model] Warning: model matvecmult
   is not calibrated, forcing calibration for this run. Use the
   STARPU_CALIBRATE environment variable to control this.
$ ...
$ STARPU_SCHED=dmda ./examples/matvecmult/matvecmult
TEST PASSED
```

Note that we force to use the scheduler `dmda` to generate performance models for the application. The application may need to be run several times before the model is calibrated.

52.3 Simulation

Then, recompile StarPU, passing --enable-simgrid to `configure`. Make sure to keep all the other `configure` options the same, and notably options such as `-enable-maxcudadev`.

```
$ ./configure --enable-simgrid
```

To specify the location of SimGrid, you can either set the environment variables `SIMGRID_CFLAGS` and `SIMGRID_LIBS`, or use the `configure` options --with-simgrid-dir, --with-simgrid-include-dir and --with-simgrid-lib-dir, for example

```
$ ./configure --with-simgrid-dir=/opt/local/simgrid
```

You can then re-run the application.

```
$ make
$ STARPU_SCHED=dmda ./examples/matvecmult/matvecmult
TEST FAILED !!!
```

It is normal that the test fails: since the computation is not actually done (that is the whole point of SimGrid), the result is wrong, of course.
If the performance model is not calibrated enough, the following error message will be displayed

```
$ STARPU_SCHED=dmda ./examples/matvecmult/matvecmult
[starpu][_starpu_load_history_based_model] Warning: model matvecmult
    is not calibrated, forcing calibration for this run. Use the
    STARPU_CALIBRATE environment variable to control this.
[starpu][_starpu_simgrid_execute_job][assert failure] Codelet
    matvecmult does not have a perfmodel, or is not calibrated enough
```

The number of devices can be chosen as usual with STARPU_NCPU, STARPU_NCUDA, and STARPU_NOPENCL, and the amount of GPU memory with STARPU_LIMIT_CUDA_MEM, STARPU_LIMIT_CUDA_devid_MEM, STARPU_LIMIT_OPENCL_MEM, and STARPU_LIMIT_OPENCL_devid_MEM.

52.4 Simulation On Another Machine

The SimGrid support even permits to perform simulations on another machine, your desktop, typically. To achieve this, one still needs to perform the Calibration step on the actual machine to be simulated, then copy them to your desktop machine (the `$STARPU_HOME/.starpu` directory). One can then perform the Simulation step on the desktop machine, by setting the environment variable STARPU_HOSTNAME to the name of the actual machine, to make StarPU use the performance models of the simulated machine even on the desktop machine. To use multiple performance models in different ranks, in case of smpi executions in a heterogeneous platform, it is possible to use the option `-hostfile-platform` in `starpu_smpirun`, that will define STARPU_MPI_HOSTNAMES with the hostnames of your hostfile.
If the desktop machine does not have CUDA or OpenCL, StarPU is still able to use SimGrid to simulate execution with CUDA/OpenCL devices, but the application source code will probably disable the CUDA and OpenCL codelets in that case. Since during SimGrid execution, the functions of the codelet are actually not called by default, one can use dummy functions such as the following to still permit CUDA or OpenCL execution.

52.5 Simulation Examples

StarPU ships a few performance models for a couple of systems: `attila`, `mirage`, `idgraf`, and `sirocco`. See Section Simulated Benchmarks for the details.

52.6 Simulations On Fake Machines

It is possible to build fake machines which do not exist, by modifying the platform file in `$STARPU_HOME/.starpu/sampling/bus/machine.platform.xml` by hand: one can add more CPUs, add GPUs (but the performance model file has to be extended as well), change the available GPU memory size, PCI memory bandwidth, etc.

52.7 Tweaking Simulation

The simulation can be tweaked, to be able to tune it between a very accurate simulation and a very simple simulation (which is thus close to scheduling theory results), see the STARPU_SIMGRID_TRANSFER_COST, STARPU_SIMGRID_CUDA_MALLOC_COST, STARPU_SIMGRID_CUDA_QUEUE_COST, STARPU_SIMGRID_TASK_SUBMIT_CO STARPU_SIMGRID_TASK_PUSH_COST, STARPU_SIMGRID_FETCHING_INPUT_COST and STARPU_SIMGRID_SCHED_COST environment variables.

52.8 MPI Applications

StarPU-MPI applications can also be run in SimGrid mode. smpi currently requires that StarPU be build statically only, so `-disable-shared` needs to be passed to `./configure`.
The application needs to be compiled with `smpicc`, and run using the `starpu_smpirun` script, for instance:

```
$ STARPU_SCHED=dmda starpu_smpirun -platform cluster.xml -hostfile hostfile ./mpi/tests/pingpong
```

Where `cluster.xml` is a SimGrid-MPI platform description, and `hostfile` the list of MPI nodes to be used. Examples of such files are available in `tools/perfmodels`. In homogeneous MPI clusters: for each MPI node, it will just replicate the architecture referred by STARPU_HOSTNAME. To use multiple performance models in different ranks, in case of a heterogeneous platform, it is possible to use the option `-hostfile-platform` in `starpu_smpirun`, that will define STARPU_MPI_HOSTNAMES with the hostnames of your hostfile.
To use FxT traces, libfxt itself also needs to be built statically, **and** with dynamic linking flags, i.e. with

```
CFLAGS=-fPIC ./configure --enable-static
```

52.9 Debugging Applications

By default, SimGrid uses its own implementation of threads, which prevents `gdb` from being able to inspect stacks of all threads. To be able to fully debug an application running with SimGrid, pass the `-cfg=contexts/factory:thread` option to the application, to make SimGrid use system threads, which `gdb` will be able to manipulate as usual.
It is also worth noting SimGrid 3.21's new parameter `-cfg=simix/breakpoint` which allows putting a breakpoint at a precise (deterministic!) timing of the execution. If for instance in an execution trace we see that something odd is happening at time 19000ms, we can use `-cfg=simix/breakpoint:19.000` and `SIGTRAP` will be raised at that point, which will thus interrupt execution within `gdb`, allowing to inspect e.g. scheduler state, etc.

52.10 Memory Usage

Since kernels are not actually run and data transfers are not actually performed, the data memory does not actually need to be allocated. This allows for instance to simulate the execution of applications processing very big data on a small laptop.
The application can for instance pass `1` (or whatever bogus pointer) to StarPU data registration functions, instead of allocating data. This will however require the application to take care of not trying to access the data, and will not work in MPI mode, which performs transfers.

Another way is to pass the STARPU_MALLOC_SIMULATION_FOLDED flag to the starpu_malloc_flags() function. An example is available in `examples/mult/xgemm.c` This will make it allocate a memory area which one can read/write, but optimized so that this does not actually consume memory. Of course, the values read from such area will be bogus, but this allows the application to keep e.g. data load, store, initialization as it is, and also work in MPI mode. A more aggressive alternative is to pass also the STARPU_MALLOC_SIMULATION_UNIQUE flag (alongside with STARPU_MALLOC_SIMULATION_FOLDED) to the starpu_malloc_flags() function. An example is available in `examples/cholesky/cholesky_tag.c` . This will make StarPU reuse the pointers for allocations of the same size without calling the folded allocation again, thus decreasing some pressure on memory management.

Note however that notably Linux kernels refuse obvious memory overcommitting by default, so a single allocation can typically not be bigger than the amount of physical memory, see `https://www.kernel.org/doc/Documentation/vm/overcommit-accounting` This prevents for instance from allocating a single huge matrix. Allocating a huge matrix in several tiles is not a problem, however. `sysctl vm.overcommit_memory=1` can also be used to allow such overcommit.

Note however that this folding is done by remapping the same file several times, and Linux kernels will also refuse to create too many memory areas. `sysctl vm.max_map_count` can be used to check and change the default (65535). By default, StarPU uses a 1MiB file, so it hopefully fits in the CPU cache. However, this limits the amount of such folded memory to a bit below 64GiB. The STARPU_MALLOC_SIMULATION_FOLD environment variable can be used to increase the size of the file.

Chapter 53

Helpers

StarPU provides several utilities functions to help programmers:

- starpu_conf_noworker() sets configuration fields so that no worker is enabled, i.e. it sets starpu_conf::ncpus to 0, starpu_conf::ncuda to 0, etc.
- starpu_is_initialized() returns a value indicating whether StarPU is already initialized, starpu_wait_initialized() only returns when the initialization is finished.
- starpu_topology_print() prints the current topology of the system, and is therefore useful for debugging purposes or for understanding the underlying architecture of the system.
- starpu_get_version() returns the version of StarPU used when running the application.
- starpu_sleep() and starpu_usleep() allow the application to pause the execution of the current thread for a specified amount of time. starpu_sleep() pauses the thread for a specified number of seconds and starpu_usleep() for a specified number of microseconds.

Chapter 54

Debugging Tools

StarPU provides several tools to help debugging applications. Execution traces can be generated and displayed graphically, see Generating Traces With FxT.

54.1 TroubleShooting In General

Generally-speaking, if you have troubles, pass --enable-debug to `configure` to enable some checks which impact performance, but will catch common issues, possibly earlier than the actual problem you are observing, which may just be a consequence of a bug that happened earlier. Also, make sure not to have the --enable-fast `configure` option, which drops very useful catchup assertions. If your program is valgrind-safe, you can use it, see Using Other Debugging Tools.

Depending on your toolchain, it might happen that you get ``undefined reference to `__stack_chk_guard'`` errors. In that case, use the `-disable-fstack-protector-all` option to avoid the issue.

Then, if your program crashes with an assertion error, a segfault, etc. you can send us the result of

```
thread apply all bt
```

run in `gdb` at the point of the crash.

In case your program just hangs, but it may also be useful in case of a crash too, it helps to source `gdbinit` as described in the next section to be able to run and send us the output of the following commands:

```
starpu-workers
starpu-tasks
starpu-print-requests
starpu-print-prequests
starpu-print-frrequests
starpu-print-irrequests
```

To give us an idea of what is happening within StarPU. If the outputs are not too long, you can even run

```
starpu-all-tasks
starpu-print-all-tasks
starpu-print-datas-summary
starpu-print-datas
```

54.2 Using The Gdb Debugger

Some `gdb` helpers are provided to show the whole StarPU state:

```
(gdb) source tools/gdbinit
(gdb) help starpu
```

For instance,

- one can print all tasks with `starpu-print-all-tasks`,
- print all data with `starpu-print-datas`,

- print all pending data transfers with `starpu-print-prequests`, `starpu-print-requests`, `starpu-print-frequests`, `starpu-print-irequests`,
- print pending MPI requests with `starpu-mpi-print-detached-requests`

Some functions can only work if --enable-debug was passed to `configure` (because they impact performance)

54.3 Using Other Debugging Tools

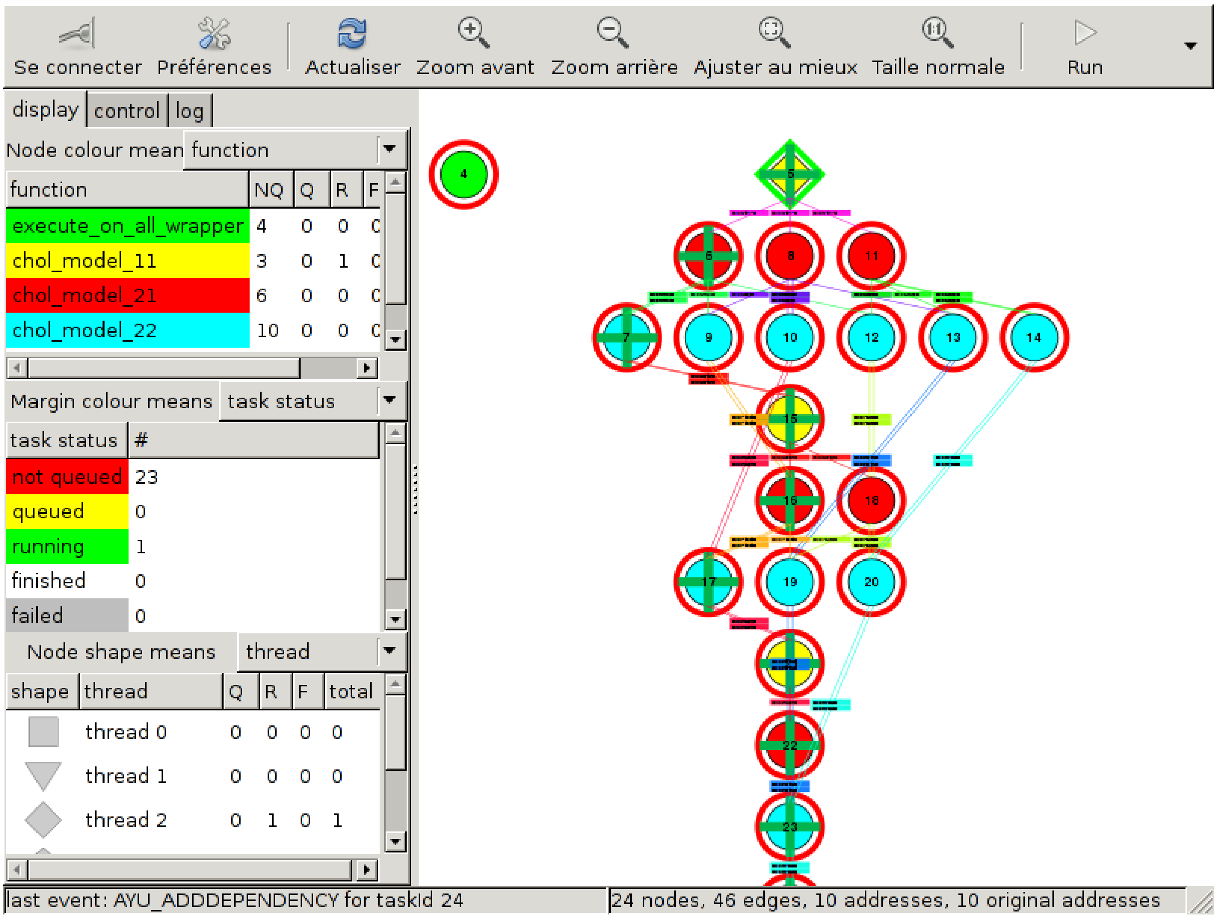
Valgrind can be used on StarPU: valgrind.h just needs to be found at `configure` time, to tell valgrind about some known false positives and disable host memory pinning. Other known false positives can be suppressed by giving the suppression files in `tools/valgrind/*.suppr` to valgrind's `--suppressions` option.
The environment variable STARPU_DISABLE_KERNELS can also be set to `1` to make StarPU does everything (schedule tasks, transfer memory, etc.) except actually calling the application-provided kernel functions, i.e. the computation will not happen. This permits to quickly check that the task scheme is working properly.

54.4 Watchdog Support

starpu_task_watchdog_set_hook() is used to set a callback function "watchdog hook" that will be called when there is no task completed during an expected time. The purpose of the watchdog hook is to allow the application to get the state for debugging.

54.5 Using The Temanejo Task Debugger

StarPU can connect to Temanejo >= 1.0rc2 (see `http://www.hlrs.de/temanejo`), to permit nice visual task debugging. To do so, build Temanejo's `libayudame.so`, install `Ayudame.h` to e.g. `/usr/local/include`, apply the `tools/patch-ayudame` to it to fix C build, re-`configure`, make sure that it found it, rebuild StarPU. Run the Temanejo GUI, give it the path to your application, any options you want to pass it, the path to `libayudame.so`.
It permits to visualize the task graph, add breakpoints, continue execution task-by-task, and run `gdb` on a given task, etc.



Node colour mean function

function	NQ	Q	R	F
🟩 execute_on_all_wrapper	4	0	0	
🟨 chol_model_11	3	0	1	
🟥 chol_model_21	6	0	0	
chol_model_22	10	0	0	

Margin colour means task status

task status	#
🟥 not queued	23
🟨 queued	0
🟩 running	1
finished	0
failed	0

Node shape means thread

shape	thread	Q	R	F	total
■	thread 0	0	0	0	0
▼	thread 1	0	0	0	0
◆	thread 2	0	1	0	1

last event: AYU_ADDDEPENDENCY for taskId 24

24 nodes, 46 edges, 10 addresses, 10 original addresses

Make sure to specify at least the same number of CPUs in the dialog box as your machine has, otherwise an error will happen during execution. Future versions of Temanejo should be able to tell StarPU the number of CPUs to use.

Tag numbers have to be below `4000000000000000000ULL` to be usable for Temanejo (to distinguish them from tasks).

Part VIII

Appendix

Chapter 55

The GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright

1. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of \`‘copyleft'', which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

2. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The \`‘Document'', below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as \`‘you''. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A \`‘Modified Version'' of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A \`‘Secondary Section'' is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

   The \`‘Invariant Sections'' are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The ‘‘Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A ‘‘Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not `Transparent'' is called`Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The ‘‘Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, ‘‘Title Page” means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The ‘‘publisher” means any person or entity that distributes copies of the Document to the public.

A section ‘‘Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as `Acknowledgements'',` Dedications", `Endorsements'', or`History".) To ‘‘Preserve the Title" of such a section when you modify the Document means that it remains a section ‘‘Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

3. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

4. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

5. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   (a) Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
   (b) List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
   (c) State on the Title page the name of the publisher of the Modified Version, as the publisher.
   (d) Preserve all the copyright notices of the Document.
   (e) Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
   (f) Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
   (g) Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
   (h) Include an unaltered copy of this License.
   (i) Preserve the section Entitled `‘History’', Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled `‘History’' in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
   (j) Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the `‘History’' section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
   (k) For any section Entitled `Acknowledgements'' or`Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
   (l) Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
   (m) Delete any section Entitled `‘Endorsements’'. Such a section may not be included in the Modified Version.
   (n) Do not retitle any existing section to be Entitled `‘Endorsements’' or to conflict in title with any Invariant Section.
   (o) Preserve any Warranty Disclaimers.

   If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled `‘Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

6. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled `‘History” in the various original documents, forming one section Entitled History''; likewise combine any sections EntitledAcknowledgements", and any sections Entitled Dedications''. You must delete all sections EntitledEndorsements."

7. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

8. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an `‘aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

9. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled Acknowledgements'', Dedications", or `‘History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

10. TERMINATION

    You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

    However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

    Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

    Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

11. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License `‘or any later version’' applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

12. RELICENSING

    `Massive Multiauthor Collaboration Site'' (or`MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A `Massive Multiauthor Collaboration'' (or`MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    `‘CC-BY-SA’' means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    `‘Incorporate’' means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is `‘eligible for relicensing’' if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

55.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (C) *year your name*. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled `‘GNU Free Documentation License’'.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the `‘with...Texts.’' line with this:

> with the Invariant Sections being *list their titles*, with the Front-Cover Texts being *list*, and with the Back-Cover Texts being *list*.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Chapter 56

Module Index

56.1 Modules

Here is a list of all modules:

Chapter 57

Module Documentation a.k.a StarPU's API

57.1 Bitmap

This is the interface for the bitmap utilities provided by StarPU.

Macros

- #define **_STARPU_LONG_BIT**
- #define **_STARPU_BITMAP_SIZE**

Functions

- static struct starpu_bitmap ∗ starpu_bitmap_create (void) STARPU_ATTRIBUTE_MALLOC
- static void starpu_bitmap_init (struct starpu_bitmap ∗b)
- static void starpu_bitmap_destroy (struct starpu_bitmap ∗b)
- static void starpu_bitmap_set (struct starpu_bitmap ∗b, int e)
- static void starpu_bitmap_unset (struct starpu_bitmap ∗b, int e)
- static void starpu_bitmap_unset_all (struct starpu_bitmap ∗b)
- static int starpu_bitmap_get (struct starpu_bitmap ∗b, int e)
- static void starpu_bitmap_unset_and (struct starpu_bitmap ∗a, struct starpu_bitmap ∗b, struct starpu_bitmap ∗c)
- static void starpu_bitmap_or (struct starpu_bitmap ∗a, struct starpu_bitmap ∗b)
- static int starpu_bitmap_and_get (struct starpu_bitmap ∗b1, struct starpu_bitmap ∗b2, int e)
- static int starpu_bitmap_cardinal (struct starpu_bitmap ∗b)
- static int starpu_bitmap_first (struct starpu_bitmap ∗b)
- static int starpu_bitmap_last (struct starpu_bitmap ∗b)
- static int starpu_bitmap_next (struct starpu_bitmap ∗b, int e)
- static int starpu_bitmap_has_next (struct starpu_bitmap ∗b, int e)

57.1.1 Detailed Description

This is the interface for the bitmap utilities provided by StarPU.

57.1.2 Function Documentation

57.1.2.1 starpu_bitmap_create()

```
static struct starpu_bitmap * starpu_bitmap_create (
             void  )  [inline], [static]
```

create a empty starpu_bitmap

57.1.2.2 starpu_bitmap_init()

```
static void starpu_bitmap_init (
             struct starpu_bitmap * b )  [inline], [static]
```

zero a starpu_bitmap

57.1.2.3 starpu_bitmap_destroy()

```
static void starpu_bitmap_destroy (
             struct starpu_bitmap * b )  [inline], [static]
```

free `b`

57.1.2.4 starpu_bitmap_set()

```
static void starpu_bitmap_set (
             struct starpu_bitmap * b,
             int e )  [inline], [static]
```

set bit `e` in `b`

57.1.2.5 starpu_bitmap_unset()

```
static void starpu_bitmap_unset (
             struct starpu_bitmap * b,
             int e )  [inline], [static]
```

unset bit `e` in `b`

57.1.2.6 starpu_bitmap_unset_all()

```
static void starpu_bitmap_unset_all (
             struct starpu_bitmap * b )  [inline], [static]
```

unset all bits in `b`

57.1.2.7 starpu_bitmap_get()

```
static int starpu_bitmap_get (
             struct starpu_bitmap * b,
             int e )  [inline], [static]
```

return true iff bit `e` is set in `b`

57.1.2.8 starpu_bitmap_unset_and()

```
static void starpu_bitmap_unset_and (
             struct starpu_bitmap * a,
             struct starpu_bitmap * b,
             struct starpu_bitmap * c )  [inline], [static]
```

Basically compute `starpu_bitmap_unset_all(a)` ; a = b & c;

57.1.2.9 starpu_bitmap_or()

```
static void starpu_bitmap_or (
             struct starpu_bitmap * a,
             struct starpu_bitmap * b )  [inline], [static]
```

Basically compute `a` |= b

57.1.2.10 starpu_bitmap_and_get()

```
static int starpu_bitmap_and_get (
             struct starpu_bitmap * b1,
             struct starpu_bitmap * b2,
             int e )  [inline], [static]
```

return 1 iff `e` is set in `b1` AND `e` is set in `b2`

57.1.2.11 starpu_bitmap_cardinal()

```
static int starpu_bitmap_cardinal (
             struct starpu_bitmap * b )  [inline], [static]
```

return the number of set bits in `b`

57.1.2.12 starpu_bitmap_first()

```
static int starpu_bitmap_first (
             struct starpu_bitmap * b )  [inline], [static]
```

return the index of the first set bit of `b`, -1 if none

57.1.2.13 starpu_bitmap_last()

```
static int starpu_bitmap_last (
             struct starpu_bitmap * b )  [inline], [static]
```

return the position of the last set bit of `b`, -1 if none

57.1.2.14 starpu_bitmap_next()

```
static int starpu_bitmap_next (
             struct starpu_bitmap * b,
             int e )  [inline], [static]
```

return the position of set bit right after `e` in `b`, -1 if none

57.1.2.15 starpu_bitmap_has_next()

```
static int starpu_bitmap_has_next (
             struct starpu_bitmap * b,
             int e )  [inline], [static]
```

todo

57.2 Hierarchical Dags

API for Hierarchical DAGS.

Macros

- #define STARPU_BUBBLE_FUNC
- #define STARPU_BUBBLE_FUNC_ARG
- #define STARPU_BUBBLE_GEN_DAG_FUNC
- #define STARPU_BUBBLE_GEN_DAG_FUNC_ARG
- #define STARPU_BUBBLE_PARENT

Typedefs

- typedef int(∗ starpu_bubble_func_t) (struct starpu_task ∗t, void ∗arg)
- typedef void(∗ starpu_bubble_gen_dag_func_t) (struct starpu_task ∗t, void ∗arg)

57.2.1 Detailed Description

API for Hierarchical DAGS.

57.2.2 Macro Definition Documentation

57.2.2.1 STARPU_BUBBLE_FUNC

`#define STARPU_BUBBLE_FUNC`

Used when calling starpu_task_insert(), must be followed by a pointer to a bubble decision function starpu_bubble_func_t

57.2.2.2 STARPU_BUBBLE_FUNC_ARG

`#define STARPU_BUBBLE_FUNC_ARG`

Used when calling starpu_task_insert(), must be followed by a pointer which will be passed to the function defined in starpu_codelet::bubble_func

57.2.2.3 STARPU_BUBBLE_GEN_DAG_FUNC

`#define STARPU_BUBBLE_GEN_DAG_FUNC`

Used when calling starpu_task_insert(), must be followed by a pointer to a bubble DAG generation function starpu_bubble_gen_dag_func_t

57.2.2.4 STARPU_BUBBLE_GEN_DAG_FUNC_ARG

`#define STARPU_BUBBLE_GEN_DAG_FUNC_ARG`

Used when calling starpu_task_insert(), must be followed by a pointer which will be passed to the function defined in starpu_codelet::bubble_gen_dag_func

57.2.2.5 STARPU_BUBBLE_PARENT

`#define STARPU_BUBBLE_PARENT`

Used when calling starpu_task_insert(), must be followed by a pointer to a task. The task will be set as the bubble parent task when using the offline tracing tool.

57.2.3 Typedef Documentation

57.2.3.1 starpu_bubble_func_t

`typedef int(* starpu_bubble_func_t) (struct starpu_task *t, void *arg)`

Bubble decision function

57.2.3.2 starpu_bubble_gen_dag_func_t

`typedef void(* starpu_bubble_gen_dag_func_t) (struct starpu_task *t, void *arg)`

Bubble DAG generation function

57.3 Codelet And Tasks

API to manipulate codelets and tasks.

Data Structures

- struct starpu_codelet
- struct starpu_data_descr
- struct starpu_task

Macros

- #define STARPU_NMAXBUFS
- #define STARPU_NOWHERE
- #define STARPU_WORKER_TO_MASK(worker_archtype)
- #define STARPU_CPU
- #define STARPU_CUDA
- #define STARPU_HIP
- #define STARPU_OPENCL
- #define STARPU_MAX_FPGA
- #define STARPU_MPI_MS
- #define STARPU_TCPIP_MS
- #define STARPU_CODELET_SIMGRID_EXECUTE
- #define STARPU_CODELET_SIMGRID_EXECUTE_AND_INJECT
- #define STARPU_CODELET_NOPLANS
- #define STARPU_CUDA_ASYNC
- #define STARPU_HIP_ASYNC
- #define STARPU_OPENCL_ASYNC
- #define STARPU_MAIN_RAM
- #define STARPU_MULTIPLE_CPU_IMPLEMENTATIONS
- #define STARPU_MULTIPLE_CUDA_IMPLEMENTATIONS
- #define STARPU_MULTIPLE_HIP_IMPLEMENTATIONS
- #define STARPU_MULTIPLE_OPENCL_IMPLEMENTATIONS
- #define STARPU_VARIABLE_NBUFFERS
- #define STARPU_SPECIFIC_NODE_LOCAL
- #define STARPU_SPECIFIC_NODE_CPU
- #define STARPU_SPECIFIC_NODE_SLOW
- #define STARPU_SPECIFIC_NODE_FAST
- #define STARPU_SPECIFIC_NODE_LOCAL_OR_CPU
- #define STARPU_SPECIFIC_NODE_NONE
- #define STARPU_TASK_TYPE_NORMAL
- #define STARPU_TASK_TYPE_INTERNAL
- #define STARPU_TASK_TYPE_DATA_ACQUIRE
- #define STARPU_TASK_INITIALIZER
- #define STARPU_TASK_GET_NBUFFERS(task)
- #define STARPU_TASK_GET_HANDLE(task, i)
- #define STARPU_TASK_GET_HANDLES(task)
- #define STARPU_TASK_SET_HANDLE(task, handle, i)
- #define STARPU_CODELET_GET_MODE(codelet, i)
- #define STARPU_CODELET_SET_MODE(codelet, mode, i)
- #define STARPU_TASK_GET_MODE(task, i)
- #define STARPU_TASK_SET_MODE(task, mode, i)
- #define STARPU_CODELET_GET_NODE(codelet, i)
- #define STARPU_CODELET_SET_NODE(codelet, __node, i)

Typedefs

- typedef void(∗ starpu_cpu_func_t) (void ∗∗, void ∗)
- typedef void(∗ starpu_cuda_func_t) (void ∗∗, void ∗)
- typedef void(∗ starpu_hip_func_t) (void ∗∗, void ∗)
- typedef void(∗ starpu_opencl_func_t) (void ∗∗, void ∗)
- typedef void(∗ starpu_max_fpga_func_t) (void ∗∗, void ∗)
- typedef struct _starpu_trs_epoch ∗ **starpu_trs_epoch_t**

Enumerations

- enum starpu_codelet_type { STARPU_SEQ , STARPU_SPMD , STARPU_FORKJOIN }
- enum starpu_task_status {
  STARPU_TASK_INIT , STARPU_TASK_INIT , STARPU_TASK_BLOCKED , STARPU_TASK_READY ,
  STARPU_TASK_RUNNING , STARPU_TASK_FINISHED , STARPU_TASK_BLOCKED_ON_TAG ,
  STARPU_TASK_BLOCKED_ON_TASK ,
  STARPU_TASK_BLOCKED_ON_DATA , STARPU_TASK_STOPPED }

Functions

- void starpu_task_init (struct starpu_task ∗task)
- void starpu_task_clean (struct starpu_task ∗task)
- struct starpu_task ∗ starpu_task_create (void) STARPU_ATTRIBUTE_MALLOC
- struct starpu_task ∗ starpu_task_create_sync (starpu_data_handle_t handle, enum starpu_data_access_mode mode) STARPU_ATTRIBUTE_MALLOC
- void starpu_task_destroy (struct starpu_task ∗task)
- void starpu_task_set_destroy (struct starpu_task ∗task)
- int starpu_task_submit (struct starpu_task ∗task)
- int starpu_task_submit_nodeps (struct starpu_task ∗task)
- int starpu_task_submit_to_ctx (struct starpu_task ∗task, unsigned sched_ctx_id)
- int starpu_task_finished (struct starpu_task ∗task)
- int starpu_task_wait (struct starpu_task ∗task)
- int starpu_task_wait_array (struct starpu_task ∗∗tasks, unsigned nb_tasks)
- int starpu_task_wait_for_all (void)
- int starpu_task_wait_for_n_submitted (unsigned n)
- int starpu_task_wait_for_all_in_ctx (unsigned sched_ctx_id)
- int starpu_task_wait_for_n_submitted_in_ctx (unsigned sched_ctx_id, unsigned n)
- int starpu_task_wait_for_no_ready (void)
- int starpu_task_nready (void)
- int starpu_task_nsubmitted (void)
- void starpu_iteration_push (unsigned long iteration)
- void starpu_iteration_pop (void)
- void starpu_do_schedule (void)
- void starpu_codelet_init (struct starpu_codelet ∗cl)
- void starpu_codelet_display_stats (struct starpu_codelet ∗cl)
- struct starpu_task ∗ starpu_task_get_current (void)
- int starpu_task_get_current_data_node (unsigned i)
- const char ∗ starpu_task_get_model_name (struct starpu_task ∗task)
- const char ∗ starpu_task_get_name (struct starpu_task ∗task)
- struct starpu_task ∗ starpu_task_dup (struct starpu_task ∗task)
- void starpu_task_set_implementation (struct starpu_task ∗task, unsigned impl)
- unsigned starpu_task_get_implementation (struct starpu_task ∗task)
- void starpu_create_sync_task (starpu_tag_t sync_tag, unsigned ndeps, starpu_tag_t ∗deps, void(∗callback)(void ∗), void ∗callback_arg)
- void starpu_create_callback_task (void(∗callback)(void ∗), void ∗callback_arg)
- void starpu_task_ft_prologue (void ∗check_ft)

- struct starpu_task ∗ starpu_task_ft_create_retry (const struct starpu_task ∗meta_task, const struct starpu_task ∗template_task, void(∗check_ft)(void ∗))
- void starpu_task_ft_failed (struct starpu_task ∗task)
- void starpu_task_ft_success (struct starpu_task ∗meta_task)
- void starpu_task_watchdog_set_hook (void(∗hook)(void ∗), void ∗hook_arg)
- char ∗ starpu_task_status_get_as_string (enum starpu_task_status status)
- void starpu_set_limit_min_submitted_tasks (int limit_min)
- void starpu_set_limit_max_submitted_tasks (int limit_min)

Variables

- struct starpu_codelet starpu_codelet_nop

57.3.1 Detailed Description

API to manipulate codelets and tasks.

57.3.2 Data Structure Documentation

57.3.2.1 struct starpu_codelet

The codelet structure describes a kernel that is possibly implemented on various targets. For compatibility, make sure to initialize the whole structure to zero, either by using explicit memset, or the function starpu_codelet_init(), or by letting the compiler implicitly do it in e.g. static storage case.
Note that the codelet structure needs to exist until the task is terminated. If dynamic codelet allocation is desired, release should be done no sooner than the starpu_task::callback_func callback time.
If the application wants to make the structure constant, it needs to be filled exactly as StarPU expects:

- starpu_codelet::cpu_funcs, starpu_codelet::cuda_funcs, etc. must be used instead of the deprecated starpu_codelet::cpu_func, starpu_codelet::cuda_func, etc.
- the starpu_codelet::where field must be set.

and additionally, starpu_codelet::checked must be set to 1 to tell StarPU that the conditions above are properly met. Also, the STARPU_CODELET_PROFILING environment variable must be set to 0. An example is provided in tests/main/const_codelet.c

Data Fields

- uint32_t where
- int(∗ can_execute)(unsigned workerid, struct starpu_task ∗task, unsigned nimpl)
- enum starpu_codelet_type type
- int max_parallelism
- starpu_cpu_func_t cpu_func
- starpu_cuda_func_t cuda_func
- starpu_opencl_func_t opencl_func
- starpu_cpu_func_t cpu_funcs [STARPU_MAXIMPLEMENTATIONS]
- starpu_cuda_func_t cuda_funcs [STARPU_MAXIMPLEMENTATIONS]
- char cuda_flags [STARPU_MAXIMPLEMENTATIONS]
- starpu_hip_func_t hip_funcs [STARPU_MAXIMPLEMENTATIONS]
- char hip_flags [STARPU_MAXIMPLEMENTATIONS]
- starpu_opencl_func_t opencl_funcs [STARPU_MAXIMPLEMENTATIONS]
- char opencl_flags [STARPU_MAXIMPLEMENTATIONS]
- starpu_max_fpga_func_t max_fpga_funcs [STARPU_MAXIMPLEMENTATIONS]
- const char ∗ cpu_funcs_name [STARPU_MAXIMPLEMENTATIONS]
- starpu_bubble_func_t bubble_func
- starpu_bubble_gen_dag_func_t bubble_gen_dag_func
- int nbuffers

- enum starpu_data_access_mode modes [STARPU_NMAXBUFS]
- enum starpu_data_access_mode ∗ dyn_modes
- unsigned specific_nodes
- int nodes [STARPU_NMAXBUFS]
- int ∗ dyn_nodes
- struct starpu_perfmodel ∗ model
- struct starpu_perfmodel ∗ energy_model
- unsigned long per_worker_stats [STARPU_NMAXWORKERS]
- const char ∗ name
- unsigned color
- void(∗ callback_func)(void ∗)
- int flags
- struct starpu_perf_counter_sample ∗ **perf_counter_sample**
- struct starpu_perf_counter_sample_cl_values ∗ **perf_counter_values**
- int checked

57.3.2.1.1 Field Documentation

57.3.2.1.1.1 where `uint32_t starpu_codelet::where`

Optional field to indicate which types of processing units are able to execute the codelet. The different values STARPU_CPU, STARPU_CUDA, STARPU_HIP, STARPU_OPENCL can be combined to specify on which types of processing units the codelet can be executed. STARPU_CPU|STARPU_CUDA for instance indicates that the codelet is implemented for both CPU cores and CUDA devices while STARPU_OPENCL indicates that it is only available on OpenCL devices. If the field is unset, its value will be automatically set based on the availability of the XXX_funcs fields defined below. It can also be set to STARPU_NOWHERE to specify that no computation has to be actually done.

57.3.2.1.1.2 can_execute `int(* starpu_codelet::can_execute) (unsigned workerid, struct starpu_task *task, unsigned nimpl)`

Define a function which should return 1 if the worker designated by `workerid` can execute the `nimpl` -th implementation of `task`, 0 otherwise.

57.3.2.1.1.3 type `enum starpu_codelet_type starpu_codelet::type`

Optional field to specify the type of the codelet. The default is STARPU_SEQ, i.e. usual sequential implementation. Other values (STARPU_SPMD or STARPU_FORKJOIN) declare that a parallel implementation is also available. See Parallel Tasks for details.

57.3.2.1.1.4 max_parallelism `int starpu_codelet::max_parallelism`

Optional field. If a parallel implementation is available, this denotes the maximum combined worker size that StarPU will use to execute parallel tasks for this codelet.

57.3.2.1.1.5 cpu_func `starpu_cpu_func_t starpu_codelet::cpu_func`

Deprecated Optional field which has been made deprecated. One should use instead the field starpu_codelet::cpu_funcs.

57.3.2.1.1.6 cuda_func `starpu_cuda_func_t starpu_codelet::cuda_func`

Deprecated Optional field which has been made deprecated. One should use instead the starpu_codelet::cuda_funcs field.

57.3.2.1.1.7 opencl_func `starpu_opencl_func_t starpu_codelet::opencl_func`

Deprecated Optional field which has been made deprecated. One should use instead the starpu_codelet::opencl_funcs field.

57.3.2.1.1.8 cpu_funcs `starpu_cpu_func_t starpu_codelet::cpu_funcs[STARPU_MAXIMPLEMENTATIONS]`

Optional array of function pointers to the CPU implementations of the codelet. The functions prototype must be:
`void cpu_func(void *buffers[], void *cl_arg)`
The first argument being the array of data managed by the data management library, and the second argument is a pointer to the argument passed from the field starpu_task::cl_arg. If the field starpu_codelet::where is set, then the field tarpu_codelet::cpu_funcs is ignored if STARPU_CPU does not appear in the field starpu_codelet::where, it must be non-`NULL` otherwise.

57.3.2.1.1.9 cuda_funcs `starpu_cuda_func_t starpu_codelet::cuda_funcs[STARPU_MAXIMPLEMENTATIONS]`

Optional array of function pointers to the CUDA implementations of the codelet. The functions must be host-functions written in the CUDA runtime API. Their prototype must be:
`void cuda_func(void *buffers[], void *cl_arg)`
If the field starpu_codelet::where is set, then the field starpu_codelet::cuda_funcs is ignored if STARPU_CUDA does not appear in the field starpu_codelet::where, it must be non-`NULL` otherwise.

57.3.2.1.1.10 cuda_flags `char starpu_codelet::cuda_flags[STARPU_MAXIMPLEMENTATIONS]`

Optional array of flags for CUDA execution. They specify some semantic details about CUDA kernel execution, such as asynchronous execution.

57.3.2.1.1.11 hip_funcs `starpu_hip_func_t starpu_codelet::hip_funcs[STARPU_MAXIMPLEMENTATIONS]`

Optional array of function pointers to the HIP implementations of the codelet. The functions must be host-functions written in the HIP runtime API. Their prototype must be:
`void hip_func(void *buffers[], void *cl_arg)`
If the field starpu_codelet::where is set, then the field starpu_codelet::hip_funcs is ignored if STARPU_HIP does not appear in the field starpu_codelet::where, it must be non-`NULL` otherwise.

57.3.2.1.1.12 hip_flags `char starpu_codelet::hip_flags[STARPU_MAXIMPLEMENTATIONS]`

Optional array of flags for HIP execution. They specify some semantic details about HIP kernel execution, such as asynchronous execution.

57.3.2.1.1.13 opencl_funcs `starpu_opencl_func_t starpu_codelet::opencl_funcs[STARPU_MAXIMPLEMENTATIONS]`

Optional array of function pointers to the OpenCL implementations of the codelet. The functions prototype must be:
`void opencl_func(void *buffers[], void *cl_arg)`
If the field starpu_codelet::where field is set, then the field starpu_codelet::opencl_funcs is ignored if STARPU_OPENCL does not appear in the field starpu_codelet::where, it must be non-`NULL` otherwise.

57.3.2.1.1.14 opencl_flags `char starpu_codelet::opencl_flags[STARPU_MAXIMPLEMENTATIONS]`

Optional array of flags for OpenCL execution. They specify some semantic details about OpenCL kernel execution, such as asynchronous execution.

57.3.2.1.1.15 max_fpga_funcs `starpu_max_fpga_func_t starpu_codelet::max_fpga_funcs[STARPU_MAXIMPLEMENTATIONS]`

Optional array of function pointers to the Maxeler FPGA implementations of the codelet. The functions prototype must be:
`void fpga_func(void *buffers[], void *cl_arg)`
The first argument being the array of data managed by the data management library, and the second argument is a pointer to the argument passed from the field starpu_task::cl_arg. If the field starpu_codelet::where is set, then the field starpu_codelet::max_fpga_funcs is ignored if STARPU_MAX_FPGA does not appear in the field starpu_codelet::where, it must be non-`NULL` otherwise.

57.3.2.1.1.16 cpu_funcs_name `const char* starpu_codelet::cpu_funcs_name[STARPU_MAXIMPLEMENTATIONS]`

Optional array of strings which provide the name of the CPU functions referenced in the array starpu_codelet::cpu_funcs. This can be used when running on MPI MS devices for StarPU to simply look up the MPI MS function implementation through its name.

57.3.2.1.1.17 bubble_func `starpu_bubble_func_t starpu_codelet::bubble_func`

Optional function to decide if the task is to be transformed into a bubble

57.3.2.1.1.18 bubble_gen_dag_func `starpu_bubble_gen_dag_func_t starpu_codelet::bubble_gen_dag_func`

Optional function to transform the task into a new graph

57.3.2.1.1.19 nbuffers `int starpu_codelet::nbuffers`

Specify the number of arguments taken by the codelet. These arguments are managed by the DSM and are accessed from the `void *buffers[]` array. The constant argument passed with the field starpu_task::cl_arg is not counted in this number. This value should not be above STARPU_NMAXBUFS. It may be set to STARPU_VARIABLE_NBUFFERS to specify that the number of buffers and their access modes will be set in starpu_task::nbuffers and starpu_task::modes or starpu_task::dyn_modes, which thus permits to define codelets with a varying number of data.

57.3.2.1.1.20 modes `enum starpu_data_access_mode starpu_codelet::modes[STARPU_NMAXBUFS]`

Is an array of starpu_data_access_mode. It describes the required access modes to the data needed by the codelet (e.g. STARPU_RW). The number of entries in this array must be specified in the field starpu_codelet::nbuffers, and should not exceed STARPU_NMAXBUFS. If insufficient, this value can be set with the configure option --enable-maxbuffers.

57.3.2.1.1.21 dyn_modes `enum starpu_data_access_mode* starpu_codelet::dyn_modes`

Is an array of starpu_data_access_mode. It describes the required access modes to the data needed by the codelet (e.g. STARPU_RW). The number of entries in this array must be specified in the field starpu_codelet::nbuffers. This field should be used for codelets having a number of data greater than STARPU_NMAXBUFS (see Setting Many Data Handles For a Task). When defining a codelet, one should either define this field or the field starpu_codelet::modes defined above.

57.3.2.1.1.22 specific_nodes `unsigned starpu_codelet::specific_nodes`

Default value is 0. If this flag is set, StarPU will not systematically send all data to the memory node where the task will be executing, it will read the starpu_codelet::nodes or starpu_codelet::dyn_nodes array to determine, for each data, on which memory node to send it.

57.3.2.1.1.23 nodes `int starpu_codelet::nodes[STARPU_NMAXBUFS]`

Optional field. When starpu_codelet::specific_nodes is 1, this specifies the memory nodes where each data should be sent to for task execution. This can be a specific memory node (>= 0), or any of STARPU_SPECIFIC_NODE_LOCAL, STARPU_SPECIFIC_NODE_CPU, STARPU_SPECIFIC_NODE_SLOW, ↩
:STARPU_SPECIFIC_NODE_FASTSTARPU_SPECIFIC_NODE_FAST, STARPU_SPECIFIC_NODE_LOCAL_OR_CPU, STARPU_SPECIFIC_NODE_NONE.
The number of entries in this array is starpu_codelet::nbuffers, and should not exceed STARPU_NMAXBUFS.

57.3.2.1.1.24 dyn_nodes `int* starpu_codelet::dyn_nodes`

Optional field. When starpu_codelet::specific_nodes is 1, this specifies the memory nodes where each data should be sent to for task execution. The number of entries in this array is starpu_codelet::nbuffers. This field should be used for codelets having a number of data greater than STARPU_NMAXBUFS (see Setting Many Data Handles For a Task). When defining a codelet, one should either define this field or the field starpu_codelet::nodes defined above.

57.3.2.1.1.25 model `struct starpu_perfmodel* starpu_codelet::model`

Optional pointer to the task duration performance model associated to this codelet. This optional field is ignored when set to `NULL` or when its field starpu_perfmodel::symbol is not set.

57.3.2.1.1.26 energy_model `struct starpu_perfmodel* starpu_codelet::energy_model`

Optional pointer to the task energy consumption performance model associated to this codelet (in J). This optional field is ignored when set to `NULL` or when its field starpu_perfmodel::symbol is not set. In the case of parallel codelets, this has to account for all processing units involved in the parallel execution.

57.3.2.1.1.27 per_worker_stats `unsigned long starpu_codelet::per_worker_stats[STARPU_NMAXWORKERS]`

Optional array for statistics collected at runtime: this is filled by StarPU and should not be accessed directly, but for example by calling the function starpu_codelet_display_stats() (See starpu_codelet_display_stats() for details).

57.3.2.1.1.28 name `const char* starpu_codelet::name`

Optional name of the codelet. This can be useful for debugging purposes.

57.3.2.1.1.29 color `unsigned starpu_codelet::color`

Optional color of the codelet. This can be useful for debugging purposes. Value 0 acts like if this field wasn't specified. Color representation is hex triplet (for example: 0xff0000 is red, 0x0000ff is blue, 0xffa500 is orange, ...).

57.3.2.1.1.30 callback_func `void(* starpu_codelet::callback_func) (void *)`

Optional field, the default value is `NULL`. This is a function pointer of prototype `void (*f)(void *)` which specifies a possible callback. If this pointer is non-`NULL`, the callback function is executed on the host after the execution of the task. If the task defines a callback, the codelet callback is not called, unless called within the task callback function. The callback is passed the value contained in the starpu_task::callback_arg field. No callback is executed if the field is set to `NULL`.

57.3.2.1.1.31 flags `int starpu_codelet::flags`

Various flags for the codelet.

57.3.2.1.1.32 checked `int starpu_codelet::checked`

Whether _starpu_codelet_check_deprecated_fields was already done or not.

57.3.2.2 struct starpu_data_descr

Describe a data handle along with an access mode.

Data Fields

starpu_data_handle_t	handle	data
enum starpu_data_access_mode	mode	access mode

57.3.2.3 struct starpu_task

Describe a task that can be offloaded on the various processing units managed by StarPU. It instantiates a codelet. It can either be allocated dynamically with the function starpu_task_create(), or declared statically. In the latter case, the programmer has to zero the structure starpu_task and to fill the different fields properly. The indicated default values correspond to the configuration of a task allocated with starpu_task_create().

Data Fields

- const char ∗ name
- const char ∗ file
- int line
- struct starpu_codelet ∗ cl
- int32_t where
- int nbuffers
- starpu_data_handle_t ∗ dyn_handles

- void ∗∗ dyn_interfaces
- enum starpu_data_access_mode ∗ dyn_modes
- starpu_data_handle_t handles [STARPU_NMAXBUFS]
- void ∗ interfaces [STARPU_NMAXBUFS]
- enum starpu_data_access_mode modes [STARPU_NMAXBUFS]
- unsigned char ∗ handles_sequential_consistency
- void ∗ cl_arg
- size_t cl_arg_size
- void ∗ cl_ret
- size_t cl_ret_size
- void(∗ epilogue_callback_func)(void ∗)
- void ∗ epilogue_callback_arg
- void(∗ callback_func)(void ∗)
- void ∗ callback_arg
- void(∗ prologue_callback_func)(void ∗)
- void ∗ prologue_callback_arg
- void(∗ prologue_callback_pop_func)(void ∗)
- void ∗ prologue_callback_pop_arg
- struct starpu_transaction ∗ transaction
- starpu_trs_epoch_t trs_epoch
- starpu_tag_t tag_id
- unsigned cl_arg_free: 1
- unsigned cl_ret_free: 1
- unsigned callback_arg_free: 1
- unsigned epilogue_callback_arg_free: 1
- unsigned prologue_callback_arg_free: 1
- unsigned prologue_callback_pop_arg_free: 1
- unsigned use_tag: 1
- unsigned sequential_consistency: 1
- unsigned synchronous: 1
- unsigned execute_on_a_specific_worker: 1
- unsigned detach: 1
- unsigned destroy: 1
- unsigned regenerate: 1
- unsigned no_submitorder: 1
- unsigned char failed
- unsigned char scheduled
- unsigned char prefetched
- unsigned workerid
- unsigned workerorder
- uint32_t ∗ workerids
- unsigned workerids_len
- int priority
- enum starpu_task_status status
- unsigned type
- unsigned color
- unsigned sched_ctx
- int hypervisor_tag
- unsigned possibly_parallel
- starpu_task_bundle_t bundle
- struct starpu_profiling_task_info ∗ profiling_info
- double flops
- double predicted
- double predicted_transfer
- double **predicted_start**

- unsigned long bubble_parent
- starpu_bubble_func_t bubble_func
- void ∗ bubble_func_arg
- starpu_bubble_gen_dag_func_t bubble_gen_dag_func
- void ∗ bubble_gen_dag_func_arg
- void ∗ sched_data

Private Attributes

- unsigned char mf_skip
- int magic
- struct starpu_task ∗ prev
- struct starpu_task ∗ next
- void ∗ starpu_private
- struct starpu_omp_task ∗ omp_task
- unsigned nb_termination_call_required

57.3.2.3.1 Field Documentation

57.3.2.3.1.1 name `const char* starpu_task::name`

Optional name of the task. This can be useful for debugging purposes.
With starpu_task_insert() and alike this can be specified thanks to STARPU_NAME followed by the const char ∗.

57.3.2.3.1.2 file `const char* starpu_task::file`

Optional file name where the task was submitted. This can be useful for debugging purposes.

57.3.2.3.1.3 line `int starpu_task::line`

Optional line number where the task was submitted. This can be useful for debugging purposes.

57.3.2.3.1.4 cl `struct starpu_codelet* starpu_task::cl`

Pointer to the corresponding structure starpu_codelet. This describes where the kernel should be executed, and supplies the appropriate implementations. When set to `NULL`, no code is executed during the tasks, such empty tasks can be useful for synchronization purposes.

57.3.2.3.1.5 where `int32_t starpu_task::where`

When set, specify where the task is allowed to be executed. When unset, take the value of starpu_codelet::where.
With starpu_task_insert() and alike this can be specified thanks to STARPU_EXECUTE_WHERE followed by an unsigned long long.

57.3.2.3.1.6 nbuffers `int starpu_task::nbuffers`

Specify the number of buffers. This is only used when starpu_codelet::nbuffers is STARPU_VARIABLE_NBUFFERS.
With starpu_task_insert() and alike this is automatically computed when using STARPU_DATA_ARRAY and alike.

57.3.2.3.1.7 dyn_handles `starpu_data_handle_t* starpu_task::dyn_handles`

Keep dyn_handles, dyn_interfaces and dyn_modes before the equivalent static arrays, so we can detect dyn_↩ handles being NULL while nbuffers being bigger that STARPU_NMAXBUFS (otherwise the overflow would put a non-NULL) Array of starpu_data_handle_t. Specify the handles to the different pieces of data accessed by the task. The number of entries in this array must be specified in the field starpu_codelet::nbuffers. This field should be used for tasks having a number of data greater than STARPU_NMAXBUFS (see Setting Many Data Handles For a Task).
When defining a task, one should either define this field or the field starpu_task::handles defined below.
With starpu_task_insert() and alike this is automatically filled when using STARPU_DATA_ARRAY and alike.

57.3.2.3.1.8 dyn_interfaces `void** starpu_task::dyn_interfaces`

Array of data pointers to the memory node where execution will happen, managed by the DSM. Is used when the field starpu_task::dyn_handles is defined.
This is filled by StarPU.

57.3.2.3.1.9 dyn_modes `enum starpu_data_access_mode* starpu_task::dyn_modes`

Used only when starpu_codelet::nbuffers is STARPU_VARIABLE_NBUFFERS. Array of starpu_data_access_mode which describes the required access modes to the data needed by the codelet (e.g. STARPU_RW). The number of entries in this array must be specified in the field starpu_codelet::nbuffers. This field should be used for codelets having a number of data greater than STARPU_NMAXBUFS (see Setting Many Data Handles For a Task). When defining a codelet, one should either define this field or the field starpu_task::modes defined below.
With starpu_task_insert() and alike this is automatically filled when using STARPU_DATA_MODE_ARRAY and alike.

57.3.2.3.1.10 handles `starpu_data_handle_t starpu_task::handles[STARPU_NMAXBUFS]`

Array of starpu_data_handle_t. Specify the handles to the different pieces of data accessed by the task. The number of entries in this array must be specified in the field starpu_codelet::nbuffers, and should not exceed STARPU_NMAXBUFS. If insufficient, this value can be set with the configure option --enable-maxbuffers.
With starpu_task_insert() and alike this is automatically filled when using STARPU_R and alike.

57.3.2.3.1.11 interfaces `void* starpu_task::interfaces[STARPU_NMAXBUFS]`

Array of Data pointers to the memory node where execution will happen, managed by the DSM.
This is filled by StarPU.

57.3.2.3.1.12 modes `enum starpu_data_access_mode starpu_task::modes[STARPU_NMAXBUFS]`

Used only when starpu_codelet::nbuffers is STARPU_VARIABLE_NBUFFERS. Array of starpu_data_access_mode which describes the required access modes to the data needed by the codelet (e.g. STARPU_RW). The number of entries in this array must be specified in the field starpu_task::nbuffers, and should not exceed STARPU_NMAXBUFS. If insufficient, this value can be set with the configure option --enable-maxbuffers.
With starpu_task_insert() and alike this is automatically filled when using STARPU_DATA_MODE_ARRAY and alike.

57.3.2.3.1.13 handles_sequential_consistency `unsigned char* starpu_task::handles_sequential_consistency`

Optional pointer to an array of characters which allows to define the sequential consistency for each handle for the current task.
With starpu_task_insert() and alike this can be specified thanks to STARPU_HANDLES_SEQUENTIAL_CONSISTENCY followed by an unsigned char $*$

57.3.2.3.1.14 cl_arg `void* starpu_task::cl_arg`

Optional pointer which is passed to the codelet through the second argument of the codelet implementation (e.g. starpu_codelet::cpu_func or starpu_codelet::cuda_func). The default value is `NULL`.
Note that the pointer is passed unchanged to most drivers, so the application has to ensure the liveness of the pointed data, by using static memory or dynamic allocation (starpu_task::cl_arg_free can be used for convenience in that case).
For the master/slave drivers however, the content pointed by cl_arg is copied to the slave, so the size of the data must be set in starpu_task::cl_arg_size.
starpu_codelet_pack_args() and starpu_codelet_unpack_args() are helpers that can can be used to respectively pack and unpack data into and from it and update starpu_task::cl_arg_size accordingly.
With starpu_task_insert() and alike this can be specified thanks to STARPU_CL_ARGS followed by a void$*$ and a size_t.

57.3.2.3.1.15 cl_arg_size `size_t starpu_task::cl_arg_size`

Optional field. For some specific drivers, the pointer starpu_task::cl_arg cannot not be directly given to the driver function. A buffer of size starpu_task::cl_arg_size needs to be allocated on the driver. This buffer is then filled with the starpu_task::cl_arg_size bytes starting at address starpu_task::cl_arg. In this case, the argument given to the codelet is therefore not the starpu_task::cl_arg pointer, but the address of the buffer in local store (LS) instead. This field is ignored for CPU, CUDA and OpenCL codelets, where the starpu_task::cl_arg pointer is given as such.
With starpu_task_insert() and alike this can be specified thanks to STARPU_CL_ARGS followed by a void$*$ and a size_t.

57.3.2.3.1.16 cl_ret `void* starpu_task::cl_ret`

Optional pointer which points to the return value of submitted task. The default value is `NULL`. starpu_codelet_pack_arg() and starpu_codelet_unpack_arg() can be used to respectively pack and unpack the return value into and form it. starpu_task::cl_ret can be used for MPI support. The only requirement is that the size of the return value must be set in starpu_task::cl_ret_size .

57.3.2.3.1.17 cl_ret_size `size_t starpu_task::cl_ret_size`

Optional field. The buffer of starpu_codelet_pack_arg() and starpu_codelet_unpack_arg() can be allocated with the starpu_task::cl_ret_size bytes starting at address starpu_task::cl_ret. starpu_task::cl_ret_size can be used for MPI support.

57.3.2.3.1.18 epilogue_callback_func `void(* starpu_task::epilogue_callback_func) (void *)`

Optional field, the default value is `NULL`. This is a function pointer of prototype `void (*f)(void *)` which specifies a possible callback. If this pointer is non-`NULL`, the callback function is executed on the host after the execution of the task. Contrary to starpu_task::callback_func, it is called before releasing tasks which depend on this task, so those cannot be already executing. The callback is passed the value contained in the starpu_task::epilogue_callback_arg field. No callback is executed if the field is set to `NULL`.
With starpu_task_insert() and alike this can be specified thanks to STARPU_EPILOGUE_CALLBACK followed by the function pointer.

57.3.2.3.1.19 epilogue_callback_arg `void* starpu_task::epilogue_callback_arg`

Optional field, the default value is `NULL`. This is the pointer passed to the epilogue callback function. This field is ignored if the field starpu_task::epilogue_callback_func is set to `NULL`.

57.3.2.3.1.20 callback_func `void(* starpu_task::callback_func) (void *)`

Optional field, the default value is `NULL`. This is a function pointer of prototype `void (*f)(void *)` which specifies a possible callback. If this pointer is non-`NULL`, the callback function is executed on the host after the execution of the task. Contrary to starpu_task::epilogue_callback, it is called after releasing tasks which depend on this task, so those might already be executing. The callback is passed the value contained in the starpu_task::callback_arg field. No callback is executed if the field is set to `NULL`.
With starpu_task_insert() and alike this can be specified thanks to STARPU_CALLBACK followed by the function pointer, or thanks to STARPU_CALLBACK_WITH_ARG (or STARPU_CALLBACK_WITH_ARG_NFREE) followed by the function pointer and the argument.

57.3.2.3.1.21 callback_arg `void* starpu_task::callback_arg`

Optional field, the default value is `NULL`. This is the pointer passed to the callback function. This field is ignored if the field starpu_task::callback_func is set to `NULL`.
With starpu_task_insert() and alike this can be specified thanks to STARPU_CALLBACK_ARG followed by the argument pointer, or thanks to STARPU_CALLBACK_WITH_ARG or STARPU_CALLBACK_WITH_ARG_NFREE followed by the function pointer and the argument.

57.3.2.3.1.22 prologue_callback_func `void(* starpu_task::prologue_callback_func) (void *)`

Optional field, the default value is `NULL`. This is a function pointer of prototype `void (*f)(void *)` which specifies a possible callback. If this pointer is non-`NULL`, the callback function is executed on the host when the task becomes ready for execution, before getting scheduled. The callback is passed the value contained in the starpu_task::prologue_callback_arg field. No callback is executed if the field is set to `NULL`.
With starpu_task_insert() and alike this can be specified thanks to STARPU_PROLOGUE_CALLBACK followed by the function pointer.

57.3.2.3.1.23 prologue_callback_arg `void* starpu_task::prologue_callback_arg`

Optional field, the default value is `NULL`. This is the pointer passed to the prologue callback function. This field is ignored if the field starpu_task::prologue_callback_func is set to `NULL`.
With starpu_task_insert() and alike this can be specified thanks to STARPU_PROLOGUE_CALLBACK_ARG followed by the argument

57.3.2.3.1.24 prologue_callback_pop_func `void(* starpu_task::prologue_callback_pop_func) (void *)`

Optional field, the default value is `NULL`. This is a function pointer of prototype `void (*f) (void*)` which specifies a possible callback. If this pointer is non-`NULL`, the callback function is executed on the host when the task is pop-ed from the scheduler, just before getting executed. The callback is passed the value contained in the starpu_task::prologue_callback_pop_arg field. No callback is executed if the field is set to `NULL`.
With starpu_task_insert() and alike this can be specified thanks to STARPU_PROLOGUE_CALLBACK_POP followed by the function pointer.

57.3.2.3.1.25 prologue_callback_pop_arg `void* starpu_task::prologue_callback_pop_arg`

Optional field, the default value is `NULL`. This is the pointer passed to the prologue_callback_pop function. This field is ignored if the field starpu_task::prologue_callback_pop_func is set to `NULL`.
With starpu_task_insert() and alike this can be specified thanks to STARPU_PROLOGUE_CALLBACK_POP_ARG followed by the argument.

57.3.2.3.1.26 transaction `struct starpu_transaction* starpu_task::transaction`

Transaction to which the task belongs, if any

57.3.2.3.1.27 trs_epoch `starpu_trs_epoch_t starpu_task::trs_epoch`

Transaction epoch to which the task belongs, if any

57.3.2.3.1.28 tag_id `starpu_tag_t starpu_task::tag_id`

Optional field. Contain the tag associated to the task if the field starpu_task::use_tag is set, ignored otherwise.
With starpu_task_insert() and alike this can be specified thanks to STARPU_TAG followed by a starpu_tag_t.

57.3.2.3.1.29 cl_arg_free `unsigned starpu_task::cl_arg_free`

Optional field. In case starpu_task::cl_arg was allocated by the application through `malloc()`, setting starpu_task::cl_arg_free to 1 makes StarPU automatically call `free(cl_arg)` when destroying the task. This saves the user from defining a callback just for that.
With starpu_task_insert() and alike this is set to 1 when using STARPU_CL_ARGS.

57.3.2.3.1.30 cl_ret_free `unsigned starpu_task::cl_ret_free`

Optional field. In case starpu_task::cl_ret was allocated by the application through `malloc()`, setting starpu_task::cl_ret_free to 1 makes StarPU automatically call `free(cl_ret)` when destroying the task.

57.3.2.3.1.31 callback_arg_free `unsigned starpu_task::callback_arg_free`

Optional field. In case starpu_task::callback_arg was allocated by the application through `malloc()`, setting starpu_task::callback_arg_free to 1 makes StarPU automatically call `free(callback_arg)` when destroying the task.
With starpu_task_insert() and alike, this is set to 1 when using STARPU_CALLBACK_ARG or STARPU_CALLBACK_WITH_ARG, or set to 0 when using STARPU_CALLBACK_ARG_NFREE

57.3.2.3.1.32 epilogue_callback_arg_free `unsigned starpu_task::epilogue_callback_arg_free`

Optional field. In case starpu_task::epilogue_callback_arg was allocated by the application through `malloc()`, setting starpu_task::epilogue_callback_arg_free to 1 makes StarPU automatically call `free(epilogue_callback_arg)` when destroying the task.

57.3.2.3.1.33 prologue_callback_arg_free `unsigned starpu_task::prologue_callback_arg_free`

Optional field. In case starpu_task::prologue_callback_arg was allocated by the application through `malloc()`, setting starpu_task::prologue_callback_arg_free to 1 makes StarPU automatically call `free(prologue_callback_arg)` when destroying the task.
With starpu_task_insert() and alike this is set to 1 when using STARPU_PROLOGUE_CALLBACK_ARG, or set to 0 when using STARPU_PROLOGUE_CALLBACK_ARG_NFREE

57.3.2.3.1.34 prologue_callback_pop_arg_free `unsigned starpu_task::prologue_callback_pop_arg_free`

Optional field. In case starpu_task::prologue_callback_pop_arg was allocated by the application through `malloc()`, setting starpu_task::prologue_callback_pop_arg_free to 1 makes StarPU automatically call `free(prologue_callback_pop_arg)` when destroying the task.
With starpu_task_insert() and alike this is set to 1 when using STARPU_PROLOGUE_CALLBACK_POP_ARG, or set to 0 when using STARPU_PROLOGUE_CALLBACK_POP_ARG_NFREE

57.3.2.3.1.35 use_tag `unsigned starpu_task::use_tag`

Optional field, the default value is 0. If set, this flag indicates that the task should be associated with the tag contained in the starpu_task::tag_id field. Tag allow the application to synchronize with the task and to express task dependencies easily.
With starpu_task_insert() and alike this is set to 1 when using STARPU_TAG.

57.3.2.3.1.36 sequential_consistency `unsigned starpu_task::sequential_consistency`

If this flag is set (which is the default), sequential consistency is enforced for the data parameters of this task for which sequential consistency is enabled. Clearing this flag permits to disable sequential consistency for this task, even if data have it enabled.
With starpu_task_insert() and alike this can be specified thanks to STARPU_SEQUENTIAL_CONSISTENCY followed by an unsigned.

57.3.2.3.1.37 synchronous `unsigned starpu_task::synchronous`

If this flag is set, the function starpu_task_submit() is blocking and returns only when the task has been executed (or if no worker is able to process the task). Otherwise, starpu_task_submit() returns immediately.
With starpu_task_insert() and alike this can be specified thanks to STARPU_TASK_SYNCHRONOUS followed an int.

57.3.2.3.1.38 execute_on_a_specific_worker `unsigned starpu_task::execute_on_a_specific_worker`

Default value is 0. If this flag is set, StarPU will bypass the scheduler and directly affect this task to the worker specified by the field starpu_task::workerid.
With starpu_task_insert() and alike this is set to 1 when using STARPU_EXECUTE_ON_WORKER.

57.3.2.3.1.39 detach `unsigned starpu_task::detach`

Optional field, default value is 1. If this flag is set, it is not possible to synchronize with the task by the means of starpu_task_wait() later on. Internal data structures are only guaranteed to be freed once starpu_task_wait() is called if the flag is not set.
With starpu_task_insert() and alike this is set to 1.

57.3.2.3.1.40 destroy `unsigned starpu_task::destroy`

Optional value. Default value is 0 for starpu_task_init(), and 1 for starpu_task_create(). If this flag is set, the task structure will automatically be freed, either after the execution of the callback if the task is detached, or during starpu_task_wait() otherwise. If this flag is not set, dynamically allocated data structures will not be freed until starpu_task_destroy() is called explicitly. Setting this flag for a statically allocated task structure will result in undefined behaviour. The flag is set to 1 when the task is created by calling starpu_task_create(). Note that starpu_task_wait_for_all() will not free any task.
With starpu_task_insert() and alike this is set to 1.
Calling starpu_task_set_destroy() can be used to set this field to 1 after submission. Indeed this function will manage concurrency against the termination of the task.

57.3.2.3.1.41 regenerate `unsigned starpu_task::regenerate`

Optional field. If this flag is set, the task will be re-submitted to StarPU once it has been executed. This flag must not be set if the flag starpu_task::destroy is set. This flag must be set before making another task depend on this one.
With starpu_task_insert() and alike this is set to 0.

57.3.2.3.1.42 no_submitorder `unsigned starpu_task::no_submitorder`

do not allocate a submitorder id for this task
With starpu_task_insert() and alike this can be specified thanks to STARPU_TASK_NO_SUBMITORDER followed by an unsigned.

57.3.2.3.1.43 mf_skip `unsigned char starpu_task::mf_skip [private]`

This is only used for tasks that use multiformat handle. This should only be used by StarPU.

57.3.2.3.1.44 failed `unsigned char starpu_task::failed`

Whether this task has failed and will thus have to be retried
Set by StarPU.

57.3.2.3.1.45 scheduled `unsigned char starpu_task::scheduled`

Whether the scheduler has pushed the task on some queue
Set by StarPU.

57.3.2.3.1.46 prefetched `unsigned char starpu_task::prefetched`

Whether the scheduler has prefetched the task's data
Set by StarPU.

57.3.2.3.1.47 workerid `unsigned starpu_task::workerid`

Optional field. If the field starpu_task::execute_on_a_specific_worker is set, this field indicates the identifier of the worker that should process this task (as returned by starpu_worker_get_id()). This field is ignored if the field starpu_task::execute_on_a_specific_worker is set to 0.
With starpu_task_insert() and alike this can be specified thanks to STARPU_EXECUTE_ON_WORKER followed by an int.

57.3.2.3.1.48 workerorder `unsigned starpu_task::workerorder`

Optional field. If the field starpu_task::execute_on_a_specific_worker is set, this field indicates the per-worker consecutive order in which tasks should be executed on the worker. Tasks will be executed in consecutive starpu_task::workerorder values, thus ignoring the availability order or task priority. See Static Scheduling for more details. This field is ignored if the field starpu_task::execute_on_a_specific_worker is set to 0.
With starpu_task_insert() and alike this can be specified thanks to STARPU_WORKER_ORDER followed by an unsigned.

57.3.2.3.1.49 workerids `uint32_t* starpu_task::workerids`

Optional field. If the field starpu_task::workerids_len is different from 0, this field indicates an array of bits (stored as uint32_t values) which indicate the set of workers which are allowed to execute the task. starpu_task::workerid takes precedence over this.
With starpu_task_insert() and alike, this can be specified along the field workerids_len thanks to STARPU_TASK_WORKERIDS followed by a number of workers and an array of bits which size is the number of workers.

57.3.2.3.1.50 workerids_len `unsigned starpu_task::workerids_len`

Optional field. This provides the number of uint32_t values in the starpu_task::workerids array.
With starpu_task_insert() and alike, this can be specified along the field workerids thanks to STARPU_TASK_WORKERIDS followed by a number of workers and an array of bits which size is the number of workers.

57.3.2.3.1.51 priority `int starpu_task::priority`

Optional field, the default value is STARPU_DEFAULT_PRIO. This field indicates a level of priority for the task. This is an integer value that must be set between the return values of the function starpu_sched_get_min_priority() for the least important tasks, and that of the function starpu_sched_get_max_priority() for the most important tasks (included). The STARPU_MIN_PRIO and STARPU_MAX_PRIO macros are provided for convenience and respectively return the value of starpu_sched_get_min_priority() and starpu_sched_get_max_priority(). Default priority is STARPU_DEFAULT_PRIO, which is always defined as 0 in order to allow static task initialization. Scheduling strategies that take priorities into account can use this parameter to take better scheduling decisions, but the scheduling policy may also ignore it.

With starpu_task_insert() and alike this can be specified thanks to STARPU_PRIORITY followed by an unsigned long long.

57.3.2.3.1.52 status `enum starpu_task_status starpu_task::status`

Current state of the task.
Call starpu_task_status_get_as_string() to get the status as a string.
Set by StarPU.

57.3.2.3.1.53 magic `int starpu_task::magic [private]`

This field is set when initializing a task. The function starpu_task_submit() will fail if the field does not have the correct value. This will hence avoid submitting tasks which have not been properly initialised.

57.3.2.3.1.54 type `unsigned starpu_task::type`

Allow to get the type of task, for filtering out tasks in profiling outputs, whether it is really internal to StarPU (STARPU_TASK_TYPE_INTERNAL), a data acquisition synchronization task (STARPU_TASK_TYPE_DATA_ACQUIRE), or a normal task (STARPU_TASK_TYPE_NORMAL)
Set by StarPU.

57.3.2.3.1.55 color `unsigned starpu_task::color`

color of the task to be used in dag.dot.
With starpu_task_insert() and alike this can be specified thanks to STARPU_TASK_COLOR followed by an int.

57.3.2.3.1.56 sched_ctx `unsigned starpu_task::sched_ctx`

Scheduling context.
With starpu_task_insert() and alike this can be specified thanks to STARPU_SCHED_CTX followed by an unsigned.

57.3.2.3.1.57 hypervisor_tag `int starpu_task::hypervisor_tag`

Help the hypervisor monitor the execution of this task.
With starpu_task_insert() and alike this can be specified thanks to STARPU_HYPERVISOR_TAG followed by an int.

57.3.2.3.1.58 possibly_parallel `unsigned starpu_task::possibly_parallel`

TODO: related with sched contexts and parallel tasks
With starpu_task_insert() and alike this can be specified thanks to STARPU_POSSIBLY_PARALLEL followed by an unsigned.

57.3.2.3.1.59 bundle `starpu_task_bundle_t starpu_task::bundle`

Optional field. The bundle that includes this task. If no bundle is used, this should be `NULL`.

57.3.2.3.1.60 profiling_info `struct starpu_profiling_task_info* starpu_task::profiling_info`

Optional field. Profiling information for the task.
With starpu_task_insert() and alike this can be specified thanks to STARPU_TASK_PROFILING_INFO followed by a pointer to the appropriate struct.

57.3.2.3.1.61 flops `double starpu_task::flops`

The application can set this to the number of floating points operations that the task will have to achieve. StarPU will measure the time that the task takes, and divide the two to get the GFlop/s achieved by the task. This will allow getting GFlops/s curves from the tool `starpu_perfmodel_plot`, and is useful for the hypervisor load balancing.
With starpu_task_insert() and alike this can be specified thanks to STARPU_FLOPS followed by a double.

57.3.2.3.1.62 predicted `double starpu_task::predicted`

Output field. Predicted duration of the task in microseconds. This field is only set if the scheduling strategy uses performance models.
Set by StarPU.

57.3.2.3.1.63 predicted_transfer `double starpu_task::predicted_transfer`

Output field. Predicted data transfer duration for the task in microseconds. This field is only valid if the scheduling strategy uses performance models.
Set by StarPU.

57.3.2.3.1.64 prev `struct starpu_task* starpu_task::prev [private]`

A pointer to the previous task. This should only be used by StarPU schedulers.

57.3.2.3.1.65 next `struct starpu_task* starpu_task::next [private]`

A pointer to the next task. This should only be used by StarPU schedulers.

57.3.2.3.1.66 starpu_private `void* starpu_task::starpu_private [private]`

This is private to StarPU, do not modify.

57.3.2.3.1.67 omp_task `struct starpu_omp_task* starpu_task::omp_task [private]`

This is private to StarPU, do not modify.

57.3.2.3.1.68 bubble_parent `unsigned long starpu_task::bubble_parent`

When using hierarchical dags, the job identifier of the bubble task which created the current task

57.3.2.3.1.69 bubble_func `starpu_bubble_func_t starpu_task::bubble_func`

When using hierarchical dags, a pointer to the bubble decision function

57.3.2.3.1.70 bubble_func_arg `void* starpu_task::bubble_func_arg`

When using hierarchical dags, a pointer to an argument to be given when calling the bubble decision function

57.3.2.3.1.71 bubble_gen_dag_func `starpu_bubble_gen_dag_func_t starpu_task::bubble_gen_dag_func`

When using hierarchical dags, a pointer to the bubble DAG generation function

57.3.2.3.1.72 bubble_gen_dag_func_arg `void* starpu_task::bubble_gen_dag_func_arg`

When using hierarchical dags, a pointer to an argument to be given when calling the bubble DAG generation function

57.3.2.3.1.73 nb_termination_call_required `unsigned starpu_task::nb_termination_call_required [private]`

This is private to StarPU, do not modify.

57.3.2.3.1.74 sched_data `void* starpu_task::sched_data`

This field is managed by the scheduler, is it allowed to do whatever with it. Typically, some area would be allocated on push, and released on pop.
With starpu_task_insert() and alike this is set when using STARPU_TASK_SCHED_DATA.

57.3.3 Macro Definition Documentation

57.3.3.1 STARPU_NMAXBUFS

```
#define STARPU_NMAXBUFS
```

Define the maximum number of buffers that tasks will be able to take as parameters. The default value is 8, it can be changed by using the configure option --enable-maxbuffers.

57.3.3.2 STARPU_NOWHERE

`#define STARPU_NOWHERE`

To be used when setting the field starpu_codelet::where to specify that the codelet has no computation part, and thus does not need to be scheduled, and data does not need to be actually loaded. This is thus essentially used for synchronization tasks.

57.3.3.3 STARPU_WORKER_TO_MASK

```
#define STARPU_WORKER_TO_MASK(
            worker_archtype )
```

Convert from enum starpu_worker_archtype to worker type mask for use in "where" fields

57.3.3.4 STARPU_CPU

`#define STARPU_CPU`

To be used when setting the field starpu_codelet::where (or starpu_task::where) to specify the codelet (or the task) may be executed on a CPU processing unit.

57.3.3.5 STARPU_CUDA

`#define STARPU_CUDA`

To be used when setting the field starpu_codelet::where (or starpu_task::where) to specify the codelet (or the task) may be executed on a CUDA processing unit.

57.3.3.6 STARPU_HIP

`#define STARPU_HIP`

To be used when setting the field starpu_codelet::where (or starpu_task::where) to specify the codelet (or the task) may be executed on a HIP processing unit.

57.3.3.7 STARPU_OPENCL

`#define STARPU_OPENCL`

To be used when setting the field starpu_codelet::where (or starpu_task::where) to specify the codelet (or the task) may be executed on a OpenCL processing unit.

57.3.3.8 STARPU_MAX_FPGA

`#define STARPU_MAX_FPGA`

To be used when setting the field starpu_codelet::where (or starpu_task::where) to specify the codelet (or the task) may be executed on a MAX FPGA.

57.3.3.9 STARPU_MPI_MS

`#define STARPU_MPI_MS`

To be used when setting the field starpu_codelet::where (or starpu_task::where) to specify the codelet (or the task) may be executed on a MPI Slave processing unit.

57.3.3.10 STARPU_TCPIP_MS

`#define STARPU_TCPIP_MS`

To be used when setting the field starpu_codelet::where (or starpu_task::where) to specify the codelet (or the task) may be executed on a TCP/IP Slave processing unit.

57.3.3.11 STARPU_CODELET_SIMGRID_EXECUTE

`#define STARPU_CODELET_SIMGRID_EXECUTE`

Value to be set in starpu_codelet::flags to execute the codelet functions even in simgrid mode.

57.3.3.12 STARPU_CODELET_SIMGRID_EXECUTE_AND_INJECT

`#define STARPU_CODELET_SIMGRID_EXECUTE_AND_INJECT`

Value to be set in starpu_codelet::flags to execute the codelet functions even in simgrid mode, and later inject the measured timing inside the simulation.

57.3.3.13 STARPU_CODELET_NOPLANS

`#define STARPU_CODELET_NOPLANS`

Value to be set in starpu_codelet::flags to make starpu_task_submit() not submit automatic asynchronous partitioning/unpartitioning.

57.3.3.14 STARPU_CUDA_ASYNC

`#define STARPU_CUDA_ASYNC`

Value to be set in starpu_codelet::cuda_flags to allow asynchronous CUDA kernel execution. This requires to use the proper CUDA stream, see CUDA-specific Optimizations

57.3.3.15 STARPU_HIP_ASYNC

`#define STARPU_HIP_ASYNC`

Value to be set in starpu_codelet::hip_flags to allow asynchronous HIP kernel execution. This requires to use the proper HIP stream

57.3.3.16 STARPU_OPENCL_ASYNC

`#define STARPU_OPENCL_ASYNC`

Value to be set in starpu_codelet::opencl_flags to allow asynchronous OpenCL kernel execution. This requires to use proper queueing, see OpenCL-specific Optimizations

57.3.3.17 STARPU_MAIN_RAM

`#define STARPU_MAIN_RAM`

To be used as memory node number for the main CPU memory node.

57.3.3.18 STARPU_MULTIPLE_CPU_IMPLEMENTATIONS

`#define STARPU_MULTIPLE_CPU_IMPLEMENTATIONS`

Deprecated Setting the field starpu_codelet::cpu_func with this macro indicates the codelet will have several implementations. The use of this macro is deprecated. One should always only define the field starpu_codelet::cpu_funcs.

57.3.3.19 STARPU_MULTIPLE_CUDA_IMPLEMENTATIONS

`#define STARPU_MULTIPLE_CUDA_IMPLEMENTATIONS`

Deprecated Setting the field starpu_codelet::cuda_func with this macro indicates the codelet will have several implementations. The use of this macro is deprecated. One should always only define the field starpu_codelet::cuda_funcs.

57.3.3.20 STARPU_MULTIPLE_HIP_IMPLEMENTATIONS

`#define STARPU_MULTIPLE_HIP_IMPLEMENTATIONS`

Deprecated Setting the field starpu_codelet::hip_func with this macro indicates the codelet will have several implementations. The use of this macro is deprecated. One should always only define the field starpu_codelet::hip_funcs.

57.3.3.21 STARPU_MULTIPLE_OPENCL_IMPLEMENTATIONS

`#define STARPU_MULTIPLE_OPENCL_IMPLEMENTATIONS`

Deprecated Setting the field starpu_codelet::opencl_func with this macro indicates the codelet will have several implementations. The use of this macro is deprecated. One should always only define the field starpu_codelet::opencl_funcs.

57.3.3.22 STARPU_VARIABLE_NBUFFERS

`#define STARPU_VARIABLE_NBUFFERS`

Value to set in starpu_codelet::nbuffers to specify that the codelet can accept a variable number of buffers, specified in starpu_task::nbuffers.

57.3.3.23 STARPU_SPECIFIC_NODE_LOCAL

`#define STARPU_SPECIFIC_NODE_LOCAL`

Value to be set in the starpu_codelet::nodes field to request StarPU to put the data in local memory of the worker running the task (this is the default behavior).

57.3.3.24 STARPU_SPECIFIC_NODE_CPU

`#define STARPU_SPECIFIC_NODE_CPU`

Value to be set in the starpu_codelet::nodes field to request StarPU to put the data in CPU-accessible memory (and let StarPU choose the NUMA node).

57.3.3.25 STARPU_SPECIFIC_NODE_SLOW

`#define STARPU_SPECIFIC_NODE_SLOW`

Value to be set in the starpu_codelet::nodes field to request StarPU to put the data in some slow memory.

57.3.3.26 STARPU_SPECIFIC_NODE_FAST

`#define STARPU_SPECIFIC_NODE_FAST`

Value to be set in the starpu_codelet::nodes field to request StarPU to put the data in some fast memory.

57.3.3.27 STARPU_SPECIFIC_NODE_LOCAL_OR_CPU

`#define STARPU_SPECIFIC_NODE_LOCAL_OR_CPU`

Value to be set in the starpu_codelet::nodes field to let StarPU decide whether to put the data in the local memory of the worker running the task, or in CPU-accessible memory (and let StarPU choose the NUMA node).

57.3.3.28 STARPU_SPECIFIC_NODE_NONE

`#define STARPU_SPECIFIC_NODE_NONE`

Value to be set in the starpu_codelet::nodes field to make StarPU not actually put the data in any particular memory, i.e. the task will only get the sequential consistency dependencies, but not actually trigger any data transfer.

57.3.3.29 STARPU_TASK_TYPE_NORMAL

`#define STARPU_TASK_TYPE_NORMAL`

To be used in the starpu_task::type field, for normal application tasks.

57.3.3.30 STARPU_TASK_TYPE_INTERNAL

`#define STARPU_TASK_TYPE_INTERNAL`

To be used in the starpu_task::type field, for StarPU-internal tasks.

57.3.3.31 STARPU_TASK_TYPE_DATA_ACQUIRE

```
#define STARPU_TASK_TYPE_DATA_ACQUIRE
```

To be used in the starpu_task::type field, for StarPU-internal data acquisition tasks.

57.3.3.32 STARPU_TASK_INITIALIZER

```
#define STARPU_TASK_INITIALIZER
```

Value to be used to initialize statically allocated tasks. This is equivalent to initializing a structure starpu_task with the function starpu_task_init().

57.3.3.33 STARPU_TASK_GET_NBUFFERS

```
#define STARPU_TASK_GET_NBUFFERS(
              task )
```

Return the number of buffers for `task`, i.e. starpu_codelet::nbuffers, or starpu_task::nbuffers if the former is STARPU_VARIABLE_NBUFFERS.

57.3.3.34 STARPU_TASK_GET_HANDLE

```
#define STARPU_TASK_GET_HANDLE(
              task,
              i )
```

Return the `i` -th data handle of `task`. If `task` is defined with a static or dynamic number of handles, will either return the `i` -th element of the field starpu_task::handles or the `i` -th element of the field starpu_task::dyn_handles (see Setting Many Data Handles For a Task)

57.3.3.35 STARPU_TASK_GET_HANDLES

```
#define STARPU_TASK_GET_HANDLES(
              task )
```

Return all the data handles of `task`. If `task` is defined with a static or dynamic number of handles, will either return all the element of the field starpu_task::handles or all the elements of the field starpu_task::dyn_handles (see Setting Many Data Handles For a Task)

57.3.3.36 STARPU_TASK_SET_HANDLE

```
#define STARPU_TASK_SET_HANDLE(
              task,
              handle,
              i )
```

Set the `i` -th data handle of `task` with `handle`. If `task` is defined with a static or dynamic number of handles, will either set the `i` -th element of the field starpu_task::handles or the `i` -th element of the field starpu_task::dyn_handles (see Setting Many Data Handles For a Task)

57.3.3.37 STARPU_CODELET_GET_MODE

```
#define STARPU_CODELET_GET_MODE(
              codelet,
              i )
```

Return the access mode of the `i` -th data handle of `codelet`. If `codelet` is defined with a static or dynamic number of handles, will either return the `i` -th element of the field starpu_codelet::modes or the `i` -th element of the field starpu_codelet::dyn_modes (see Setting Many Data Handles For a Task)

57.3.3.38 STARPU_CODELET_SET_MODE

```
#define STARPU_CODELET_SET_MODE(
              codelet,
              mode,
              i )
```

Set the access mode of the `i` -th data handle of `codelet`. If `codelet` is defined with a static or dynamic number of handles, will either set the `i` -th element of the field starpu_codelet::modes or the `i` -th element of the field starpu_codelet::dyn_modes (see Setting Many Data Handles For a Task)

57.3.3.39 STARPU_TASK_GET_MODE

```
#define STARPU_TASK_GET_MODE(
                task,
                i )
```

Return the access mode of the `i` -th data handle of `task`. If `task` is defined with a static or dynamic number of handles, will either return the `i` -th element of the field starpu_task::modes or the `i` -th element of the field starpu_task::dyn_modes (see Setting Many Data Handles For a Task)

57.3.3.40 STARPU_TASK_SET_MODE

```
#define STARPU_TASK_SET_MODE(
                task,
                mode,
                i )
```

Set the access mode of the `i` -th data handle of `task`. If `task` is defined with a static or dynamic number of handles, will either set the `i` -th element of the field starpu_task::modes or the `i` -th element of the field starpu_task::dyn_modes (see Setting Many Data Handles For a Task)

57.3.3.41 STARPU_CODELET_GET_NODE

```
#define STARPU_CODELET_GET_NODE(
                codelet,
                i )
```

Return the target node of the `i` -th data handle of `codelet`. If `node` is defined with a static or dynamic number of handles, will either return the `i` -th element of the field starpu_codelet::nodes or the `i` -th element of the field starpu_codelet::dyn_nodes (see Setting Many Data Handles For a Task)

57.3.3.42 STARPU_CODELET_SET_NODE

```
#define STARPU_CODELET_SET_NODE(
                codelet,
                __node,
                i )
```

Set the target node of the `i` -th data handle of `codelet`. If `codelet` is defined with a static or dynamic number of handles, will either set the `i` -th element of the field starpu_codelet::nodes or the `i` -th element of the field starpu_codelet::dyn_nodes (see Setting Many Data Handles For a Task)

57.3.4 Typedef Documentation

57.3.4.1 starpu_cpu_func_t

```
typedef void(* starpu_cpu_func_t) (void **, void *)
```

CPU implementation of a codelet.

57.3.4.2 starpu_cuda_func_t

```
typedef void(* starpu_cuda_func_t) (void **, void *)
```

CUDA implementation of a codelet.

57.3.4.3 starpu_hip_func_t

```
typedef void(* starpu_hip_func_t) (void **, void *)
```

HIP implementation of a codelet.

57.3.4.4 starpu_opencl_func_t

`typedef void(* starpu_opencl_func_t) (void **, void *)`

OpenCL implementation of a codelet.

57.3.4.5 starpu_max_fpga_func_t

`typedef void(* starpu_max_fpga_func_t) (void **, void *)`

Maxeler FPGA implementation of a codelet.

57.3.5 Enumeration Type Documentation

57.3.5.1 starpu_codelet_type

`enum starpu_codelet_type`

Describe the type of parallel task. See Parallel Tasks for details.

Enumerator

STARPU_SEQ	(default) for classical sequential tasks.
STARPU_SPMD	for a parallel task whose threads are handled by StarPU, the code has to use starpu_combined_worker_get_size() and starpu_combined_worker_get_rank() to distribute the work.
STARPU_FORKJOIN	for a parallel task whose threads are started by the codelet function, which has to use starpu_combined_worker_get_size() to determine how many threads should be started.

57.3.5.2 starpu_task_status

`enum starpu_task_status`

todo

Enumerator

STARPU_TASK_INIT	The task has just been initialized.
STARPU_TASK_INIT	The task has just been initialized.
STARPU_TASK_BLOCKED	The task has just been submitted, and its dependencies has not been checked yet.
STARPU_TASK_READY	The task is ready for execution.
STARPU_TASK_RUNNING	The task is running on some worker.
STARPU_TASK_FINISHED	The task is finished executing.
STARPU_TASK_BLOCKED_ON_TAG	The task is waiting for a tag.
STARPU_TASK_BLOCKED_ON_TASK	The task is waiting for a task.
STARPU_TASK_BLOCKED_ON_DATA	The task is waiting for some data.
STARPU_TASK_STOPPED	The task is stopped.

57.3.6 Function Documentation

57.3.6.1 starpu_task_init()

```
void starpu_task_init (
            struct starpu_task * task )
```

Initialize `task` with default values. This function is implicitly called by starpu_task_create(). By default, tasks initialized with starpu_task_init() must be deinitialized explicitly with starpu_task_clean(). Tasks can also be initialized statically, using STARPU_TASK_INITIALIZER. See Performance Model Calibration for more details.

57.3.6.2 starpu_task_clean()

```
void starpu_task_clean (
            struct starpu_task * task )
```

Release all the structures automatically allocated to execute `task`, but not the task structure itself and values set by the user remain unchanged. It is thus useful for statically allocated tasks for instance. It is also useful when users want to execute the same operation several times with as least overhead as possible. It is called automatically by starpu_task_destroy(). It has to be called only after explicitly waiting for the task or after starpu_shutdown() (waiting for the callback is not enough, since StarPU still manipulates the task after calling the callback). See Performance Model Calibration for more details.

57.3.6.3 starpu_task_create()

```
struct starpu_task * starpu_task_create (
            void )
```

Allocate a task structure and initialize it with default values. Tasks allocated dynamically with starpu_task_create() are automatically freed when the task is terminated. This means that the task pointer can not be used any more once the task is submitted, since it can be executed at any time (unless dependencies make it wait) and thus freed at any time. If the field starpu_task::destroy is explicitly unset, the resources used by the task have to be freed by calling starpu_task_destroy(). See Submitting A Task for more details.

57.3.6.4 starpu_task_create_sync()

```
struct starpu_task * starpu_task_create_sync (
            starpu_data_handle_t handle,
            enum starpu_data_access_mode mode )
```

Allocate a task structure that does nothing but accesses data `handle` with mode `mode`. This allows to synchronize with the task graph, according to the sequential consistency, against tasks submitted before or after submitting this task. One can then use starpu_task_declare_deps_array() or starpu_task_end_dep_add() / starpu_task_end_dep_release() to add dependencies against this task before submitting it. See Synchronization Tasks for more details.

57.3.6.5 starpu_task_destroy()

```
void starpu_task_destroy (
            struct starpu_task * task )
```

Free the resource allocated during starpu_task_create() and associated with `task`. This function is called automatically after the execution of a task when the field starpu_task::destroy is set, which is the default for tasks created by starpu_task_create(). Calling this function on a statically allocated task results in an undefined behaviour. See Per-task Feedback and Performance Model Example for more details.

57.3.6.6 starpu_task_set_destroy()

```
void starpu_task_set_destroy (
            struct starpu_task * task )
```

Tell StarPU to free the resources associated with `task` when the task is over. This is equivalent to having set task->destroy = 1 before submission, the difference is that this can be called after submission and properly deals with concurrency with the task execution. See Waiting For Tasks for more details.

57.3.6.7 starpu_task_submit()

```
int starpu_task_submit (
            struct starpu_task * task )
```

Submit `task` to StarPU. Calling this function does not mean that the task will be executed immediately as there can be data or task (tag) dependencies that are not fulfilled yet: StarPU will take care of scheduling this task with respect to such dependencies. This function returns immediately if the field starpu_task::synchronous is set to 0, and block until the termination of the task otherwise. It is also possible to synchronize the application with asynchronous tasks by the means of tags, using the function starpu_tag_wait() function for instance. In case of success, this function returns 0, a return value of `-ENODEV` means that there is no worker able to process this task (e.g. there is no GPU available and this task is only implemented for CUDA devices). starpu_task_submit() can be called from anywhere, including codelet functions and callbacks, provided that the field starpu_task::synchronous is set to 0. See Submitting A Task for more details.

57.3.6.8 starpu_task_submit_nodeps()

```
int starpu_task_submit_nodeps (
            struct starpu_task * task )
```

Submit `task` to StarPU with dependency bypass.

This can only be called on behalf of another task which has already taken the proper dependencies, e.g. this task is just an attempt of doing the actual computation of that task. See Retrying tasks for more details.

57.3.6.9 starpu_task_submit_to_ctx()

```
int starpu_task_submit_to_ctx (
            struct starpu_task * task,
            unsigned sched_ctx_id )
```

Submit `task` to the context `sched_ctx_id`. By default, starpu_task_submit() submits the task to a global context that is created automatically by StarPU. See Submitting Tasks To A Context for more details.

57.3.6.10 starpu_task_finished()

```
int starpu_task_finished (
            struct starpu_task * task )
```

Return 1 if `task` is terminated. See Waiting For Tasks for more details.

57.3.6.11 starpu_task_wait()

```
int starpu_task_wait (
            struct starpu_task * task )
```

Block until `task` has been executed. It is not possible to synchronize with a task more than once. It is not possible to wait for synchronous or detached tasks. Upon successful completion, this function returns 0. Otherwise, `-EINVAL` indicates that the specified task was either synchronous or detached. See Submitting A Task for more details.

57.3.6.12 starpu_task_wait_array()

```
int starpu_task_wait_array (
            struct starpu_task ** tasks,
            unsigned nb_tasks )
```

Allow to wait for an array of tasks. Upon successful completion, this function returns 0. Otherwise, `-EINVAL` indicates that one of the tasks was either synchronous or detached. See Waiting For Tasks for more details.

57.3.6.13 starpu_task_wait_for_all()

```
int starpu_task_wait_for_all (
            void )
```

Block until all the tasks that were submitted (to the current context or the global one if there is no current context) are terminated. It does not destroy these tasks. See Submitting A Task for more details.

57.3.6.14 starpu_task_wait_for_n_submitted()

```
int starpu_task_wait_for_n_submitted (
            unsigned n )
```

Block until there are `n` submitted tasks left (to the current context or the global one if there is no current context) to be executed. It does not destroy these tasks. See How To Reuse Memory for more details.

57.3.6.15 starpu_task_wait_for_all_in_ctx()

```
int starpu_task_wait_for_all_in_ctx (
            unsigned sched_ctx_id )
```

Wait until all the tasks that were already submitted to the context `sched_ctx_id` have been terminated. See Waiting For Tasks for more details.

57.3.6.16 starpu_task_wait_for_n_submitted_in_ctx()

```
int starpu_task_wait_for_n_submitted_in_ctx (
            unsigned sched_ctx_id,
            unsigned n )
```

Wait until there are `n` tasks submitted left to be executed that were already submitted to the context `sched_ctx_id`. See Waiting For Tasks for more details.

57.3.6.17 starpu_task_wait_for_no_ready()

```
int starpu_task_wait_for_no_ready (
            void )
```

Wait until there is no more ready task. See Waiting For Tasks for more details.

57.3.6.18 starpu_task_nready()

```
int starpu_task_nready (
            void )
```

Return the number of submitted tasks which are ready for execution are already executing. It thus does not include tasks waiting for dependencies. See Waiting For Tasks for more details.

57.3.6.19 starpu_task_nsubmitted()

```
int starpu_task_nsubmitted (
            void )
```

Return the number of submitted tasks which have not completed yet. See Waiting For Tasks for more details.

57.3.6.20 starpu_iteration_push()

```
void starpu_iteration_push (
            unsigned long iteration )
```

Set the iteration number for all the tasks to be submitted after this call. This is typically called at the beginning of a task submission loop. This number will then show up in tracing tools. A corresponding starpu_iteration_pop() call must be made to match the call to starpu_iteration_push(), at the end of the same task submission loop, typically.
Nested calls to starpu_iteration_push() and starpu_iteration_pop() are allowed, to describe a loop nest for instance, provided that they match properly.
See Creating a Gantt Diagram for more details.

57.3.6.21 starpu_iteration_pop()

```
void starpu_iteration_pop (
            void )
```

Drop the iteration number for submitted tasks. This must match a previous call to starpu_iteration_push(), and is typically called at the end of a task submission loop. See Creating a Gantt Diagram for more details.

57.3.6.22 starpu_do_schedule()

```
void starpu_do_schedule (
             void )
```

See Graph-based Scheduling for more details.

57.3.6.23 starpu_codelet_init()

```
void starpu_codelet_init (
             struct starpu_codelet * cl )
```

Initialize `cl` with default values. Codelets should preferably be initialized statically as shown in Defining A Codelet. However such a initialisation is not always possible, e.g. when using C++. See Defining A Codelet for more details.

57.3.6.24 starpu_codelet_display_stats()

```
void starpu_codelet_display_stats (
             struct starpu_codelet * cl )
```

Output on `stderr` some statistics on the codelet `cl`. See Per-codelet Feedback for more details.

57.3.6.25 starpu_task_get_current()

```
struct starpu_task * starpu_task_get_current (
             void )
```

Return the task currently executed by the worker, or `NULL` if it is called either from a thread that is not a task or simply because there is no task being executed at the moment. See Per-task Feedback for more details.

57.3.6.26 starpu_task_get_current_data_node()

```
int starpu_task_get_current_data_node (
             unsigned i )
```

Return the memory node number of parameter `i` of the task currently executed, or -1 if it is called either from a thread that is not a task or simply because there is no task being executed at the moment.
Usually, the returned memory node number is simply the memory node for the current worker. That may however be different when using e.g. starpu_codelet::specific_nodes.
See Specifying A Target Node For Task Data for more details.

57.3.6.27 starpu_task_get_model_name()

```
const char * starpu_task_get_model_name (
             struct starpu_task * task )
```

Return the name of the performance model of `task`. See Performance Model Example for more details.

57.3.6.28 starpu_task_get_name()

```
const char * starpu_task_get_name (
             struct starpu_task * task )
```

Return the name of `task`, i.e. either its starpu_task::name field, or the name of the corresponding performance model. See Getting Task Details for more details.

57.3.6.29 starpu_task_dup()

```
struct starpu_task * starpu_task_dup (
             struct starpu_task * task )
```

Allocate a task structure which is the exact duplicate of `task`. See Other Task Utility Functions for more details.

57.3.6.30 starpu_task_set_implementation()

```
void starpu_task_set_implementation (
            struct starpu_task * task,
            unsigned impl )
```

This function should be called by schedulers to specify the codelet implementation to be executed when executing `task`. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.3.6.31 starpu_task_get_implementation()

```
unsigned starpu_task_get_implementation (
            struct starpu_task * task )
```

Return the codelet implementation to be executed when executing `task`. See Helper functions for defining a scheduling policy (Basic
for more details.

57.3.6.32 starpu_create_sync_task()

```
void starpu_create_sync_task (
            starpu_tag_t sync_tag,
            unsigned ndeps,
            starpu_tag_t * deps,
            void(*)(void *) callback,
            void * callback_arg )
```

Create and submit an empty task that unlocks a tag once all its dependencies are fulfilled. See Synchronization Tasks for more details.

57.3.6.33 starpu_create_callback_task()

```
void starpu_create_callback_task (
            void(*)(void *) callback,
            void * callback_arg )
```

Create and submit an empty task with the given callback. See Synchronization Tasks for more details.

57.3.6.34 starpu_task_ft_prologue()

```
void starpu_task_ft_prologue (
            void * check_ft )
```

Function to be used as a prologue callback to enable fault tolerance for the task. This prologue will create a try-task, i.e a duplicate of the task, which will to the actual computation.

The prologue argument can be set to a check_ft function that will be called on termination of the duplicate, which can check the result of the task, and either confirm success, or resubmit another attempt. If it is not set, the default implementation is to just resubmit a new try-task.

See Retrying tasks for more details.

57.3.6.35 starpu_task_ft_create_retry()

```
struct starpu_task * starpu_task_ft_create_retry (
            const struct starpu_task * meta_task,
            const struct starpu_task * template_task,
            void(*)(void *) check_ft )
```

Create a try-task for a `meta_task`, given a `template_task` task template. The meta task can be passed as template on the first call, but since it is mangled by starpu_task_ft_create_retry(), further calls (typically made by the check_ft callback) need to be passed the previous try-task as template task.

`check_ft` is similar to the prologue argument of starpu_task_ft_prologue(), and is typically set to the very function calling starpu_task_ft_create_retry().

The try-task is returned, and can be modified (e.g. to change scheduling parameters) before being submitted with starpu_task_submit_nodeps().

See Retrying tasks for more details.

57.3.6.36 starpu_task_ft_failed()

```
void starpu_task_ft_failed (
            struct starpu_task * task )
```

Record that this task failed, and should thus be retried. This is usually called from the task codelet function itself, after checking the result and noticing that the computation went wrong, and thus the task should be retried. The performance of this task execution will not be recorded for performance models.
This can only be called for a task whose data access modes are either STARPU_R and STARPU_W.

57.3.6.37 starpu_task_ft_success()

```
void starpu_task_ft_success (
            struct starpu_task * meta_task )
```

Notify that the try-task was successful and thus the meta-task was successful. See Retrying tasks for more details.

57.3.6.38 starpu_task_watchdog_set_hook()

```
void starpu_task_watchdog_set_hook (
            void(*)(void *) hook,
            void * hook_arg )
```

Set the function to call when the watchdog detects that StarPU has not finished any task for STARPU_WATCHDOG_TIMEOUT seconds. See Watchdog Support for more details.

57.3.6.39 starpu_task_status_get_as_string()

```
char * starpu_task_status_get_as_string (
            enum starpu_task_status status )
```

Return the given status as a string

57.3.6.40 starpu_set_limit_min_submitted_tasks()

```
void starpu_set_limit_min_submitted_tasks (
            int limit_min )
```

Specify a minimum number of submitted tasks allowed at a given time, this allows to control the task submission flow. The value can also be specified with the environment variable STARPU_LIMIT_MIN_SUBMITTED_TASKS. See How To Reduce The Memory Footprint Of Internal Data Structures for more details.

57.3.6.41 starpu_set_limit_max_submitted_tasks()

```
void starpu_set_limit_max_submitted_tasks (
            int limit_min )
```

Specify a maximum number of submitted tasks allowed at a given time, this allows to control the task submission flow. The value can also be specified with the environment variable STARPU_LIMIT_MAX_SUBMITTED_TASKS. See How To Reduce The Memory Footprint Of Internal Data Structures for more details.

57.3.7 Variable Documentation

57.3.7.1 starpu_codelet_nop

```
struct starpu_codelet starpu_codelet_nop [extern]
```

Codelet with empty function defined for all drivers

57.4 CUDA Extensions

Macros

- #define STARPU_USE_CUDA
- #define STARPU_HAVE_NVML_H
- #define STARPU_MAXCUDADEVS
- #define STARPU_CUBLAS_REPORT_ERROR(status)
- #define STARPU_CUDA_REPORT_ERROR(status)

Functions

- void starpu_cublas_report_error (const char ∗func, const char ∗file, int line, int status)
- void starpu_cuda_report_error (const char ∗func, const char ∗file, int line, cudaError_t status)
- cudaStream_t starpu_cuda_get_local_stream (void)
- const struct cudaDeviceProp ∗ starpu_cuda_get_device_properties (unsigned workerid)
- int starpu_cuda_copy_async_sync (void ∗src_ptr, unsigned src_node, void ∗dst_ptr, unsigned dst_node, size_t ssize, cudaStream_t stream, enum cudaMemcpyKind kind)
- int starpu_cuda_copy2d_async_sync (void ∗src_ptr, unsigned src_node, void ∗dst_ptr, unsigned dst_node, size_t blocksize, size_t numblocks, size_t ld_src, size_t ld_dst, cudaStream_t stream, enum cudaMemcpy↩ Kind kind)
- int starpu_cuda_copy3d_async_sync (void ∗src_ptr, unsigned src_node, void ∗dst_ptr, unsigned dst_node, size_t blocksize, size_t numblocks_1, size_t ld1_src, size_t ld1_dst, size_t numblocks_2, size_t ld2_src, size_t ld2_dst, cudaStream_t stream, enum cudaMemcpyKind kind)
- void starpu_cuda_set_device (unsigned devid)
- nvmlDevice_t starpu_cuda_get_nvmldev (unsigned devid)

- void starpu_cusparse_init (void)
- void starpu_cusparse_shutdown (void)
- cusparseHandle_t starpu_cusparse_get_local_handle (void)

- void starpu_cublas_init (void)
- void starpu_cublas_set_stream (void)
- void starpu_cublas_shutdown (void)

- cublasHandle_t starpu_cublas_get_local_handle (void)

- void starpu_cusolver_init (void)

57.4.1 Detailed Description

57.4.2 Macro Definition Documentation

57.4.2.1 STARPU_USE_CUDA

`#define STARPU_USE_CUDA`

Defined when StarPU has been installed with CUDA support. It should be used in your code to detect the availability of CUDA.

57.4.2.2 STARPU_HAVE_NVML_H

`#define STARPU_HAVE_NVML_H`

Defined when StarPU has been installed with NVidia-ML support. It should be used in your code to detect the availability of NVML-related functions.

57.4.2.3 STARPU_MAXCUDADEVS

```
#define STARPU_MAXCUDADEVS
```

Define the maximum number of CUDA devices that are supported by StarPU.

57.4.2.4 STARPU_CUBLAS_REPORT_ERROR

```
#define STARPU_CUBLAS_REPORT_ERROR(
              status )
```

Call starpu_cublas_report_error(), passing the current function, file and line position.

57.4.2.5 STARPU_CUDA_REPORT_ERROR

```
#define STARPU_CUDA_REPORT_ERROR(
              status )
```

Call starpu_cuda_report_error(), passing the current function, file and line position.

57.4.3 Function Documentation

57.4.3.1 starpu_cublas_report_error()

```
void starpu_cublas_report_error (
             const char * func,
             const char * file,
             int line,
             int status )
```

Report a CUBLAS error. See CUDA Support for more details.

57.4.3.2 starpu_cuda_report_error()

```
void starpu_cuda_report_error (
             const char * func,
             const char * file,
             int line,
             cudaError_t status )
```

Report a CUDA error. See CUDA Support for more details.

57.4.3.3 starpu_cuda_get_local_stream()

```
cudaStream_t starpu_cuda_get_local_stream (
             void )
```

Return the current worker's CUDA stream. StarPU provides a stream for every CUDA device controlled by StarPU. This function is only provided for convenience so that programmers can easily use asynchronous operations within codelets without having to create a stream by hand. Note that the application is not forced to use the stream provided by starpu_cuda_get_local_stream() and may also create its own streams. Synchronizing with `cudaDeviceSynchronize()` is allowed, but will reduce the likelihood of having all transfers overlapped. See CUDA-specific Optimizations for more details.

57.4.3.4 starpu_cuda_get_device_properties()

```
const struct cudaDeviceProp * starpu_cuda_get_device_properties (
             unsigned workerid )
```

Return a pointer to device properties for worker `workerid` (assumed to be a CUDA worker). See Enabling Implementation According To Capabilities for more details.

57.4.3.5 starpu_cuda_copy_async_sync()

```
int starpu_cuda_copy_async_sync (
            void * src_ptr,
            unsigned src_node,
            void * dst_ptr,
            unsigned dst_node,
            size_t ssize,
            cudaStream_t stream,
            enum cudaMemcpyKind kind )
```

Copy `ssize` bytes from the pointer `src_ptr` on `src_node` to the pointer `dst_ptr` on `dst_node`. The function first tries to copy the data asynchronous (unless `stream` is `NULL`). If the asynchronous copy fails or if `stream` is `NULL`, it copies the data synchronously. The function returns `-EAGAIN` if the asynchronous launch was successful. It returns 0 if the synchronous copy was successful, or fails otherwise.
See CUDA Support for more details.

57.4.3.6 starpu_cuda_copy2d_async_sync()

```
int starpu_cuda_copy2d_async_sync (
            void * src_ptr,
            unsigned src_node,
            void * dst_ptr,
            unsigned dst_node,
            size_t blocksize,
            size_t numblocks,
            size_t ld_src,
            size_t ld_dst,
            cudaStream_t stream,
            enum cudaMemcpyKind kind )
```

Copy `numblocks` blocks of `blocksize` bytes from the pointer `src_ptr` on `src_node` to the pointer `dst_ptr` on `dst_node`.
The blocks start at addresses which are ld_src (resp. ld_dst) bytes apart in the source (resp. destination) interface.
The function first tries to copy the data asynchronous (unless `stream` is `NULL`). If the asynchronous copy fails or if `stream` is `NULL`, it copies the data synchronously. The function returns `-EAGAIN` if the asynchronous launch was successful. It returns 0 if the synchronous copy was successful, or fails otherwise.
See CUDA Support for more details.

57.4.3.7 starpu_cuda_copy3d_async_sync()

```
int starpu_cuda_copy3d_async_sync (
            void * src_ptr,
            unsigned src_node,
            void * dst_ptr,
            unsigned dst_node,
            size_t blocksize,
            size_t numblocks_1,
            size_t ld1_src,
            size_t ld1_dst,
            size_t numblocks_2,
            size_t ld2_src,
            size_t ld2_dst,
            cudaStream_t stream,
            enum cudaMemcpyKind kind )
```

Copy `numblocks_1 * numblocks_2` blocks of `blocksize` bytes from the pointer `src_ptr` on `src_node` to the pointer `dst_ptr` on `dst_node`.
The blocks are grouped by `numblocks_1` blocks whose start addresses are ld1_src (resp. ld1_dst) bytes apart in the source (resp. destination) interface.
The function first tries to copy the data asynchronous (unless `stream` is `NULL`). If the asynchronous copy fails or if `stream` is `NULL`, it copies the data synchronously. The function returns `-EAGAIN` if the asynchronous launch

was successful. It returns 0 if the synchronous copy was successful, or fails otherwise.
See CUDA Support for more details.

57.4.3.8 starpu_cuda_set_device()

```
void starpu_cuda_set_device (
            unsigned devid )
```

Call `cudaSetDevice(devid)` or `cudaGLSetGLDevice(devid)`, according to whether `devid` is among the field starpu_conf::cuda_opengl_interoperability.
See CUDA Support for more details.

57.4.3.9 starpu_cuda_get_nvmldev()

```
nvmlDevice_t starpu_cuda_get_nvmldev (
            unsigned devid )
```

Return the nvml device for a CUDA device See CUDA Support for more details.

57.4.3.10 starpu_cusparse_init()

```
void starpu_cusparse_init (
            void )
```

Initialize CUSPARSE on every CUDA device controlled by StarPU. This call blocks until CUSPARSE has been properly initialized on every device. See CUDA-specific Optimizations for more details.

57.4.3.11 starpu_cublas_init()

```
void starpu_cublas_init (
            void )
```

Initialize CUBLAS on every CUDA device. The CUBLAS library must be initialized prior to any CUBLAS call. Calling starpu_cublas_init() will initialize CUBLAS on every CUDA device controlled by StarPU. This call blocks until CUBLAS has been properly initialized on every device. See CUDA-specific Optimizations for more details.

57.4.3.12 starpu_cublas_get_local_handle()

```
cublasHandle_t starpu_cublas_get_local_handle (
            void )
```

Return the CUBLAS handle to be used to queue CUBLAS kernels. It is properly initialized and configured for multistream by starpu_cublas_init(). See CUDA-specific Optimizations for more details.

57.4.3.13 starpu_cusolver_init()

```
void starpu_cusolver_init (
            void )
```

Initialize CUSOLVER on every CUDA device controlled by StarPU. This call blocks until CUSOLVER has been properly initialized on every device.
See CUDA-specific Optimizations

57.4.3.14 starpu_cublas_set_stream()

```
void starpu_cublas_set_stream (
            void )
```

Set the proper CUBLAS stream for CUBLAS v1. This must be called from the CUDA codelet before calling CUBLAS v1 kernels, so that they are queued on the proper CUDA stream. When using one thread per CUDA worker, this function does not do anything since the CUBLAS stream does not change, and is set once by starpu_cublas_init().
See CUDA-specific Optimizations for more details.

57.4.3.15 starpu_cublas_shutdown()

```
void starpu_cublas_shutdown (
            void )
```

Synchronously deinitialize the CUBLAS library on every CUDA device. See CUDA-specific Optimizations for more details.

57.4.3.16 starpu_cusparse_shutdown()

```
void starpu_cusparse_shutdown (
            void )
```

Synchronously deinitialize the CUSPARSE library on every CUDA device. See CUDA-specific Optimizations for more details.

57.4.3.17 starpu_cusparse_get_local_handle()

```
cusparseHandle_t starpu_cusparse_get_local_handle (
            void )
```

Return the CUSPARSE handle to be used to queue CUSPARSE kernels. It is properly initialized and configured for multistream by starpu_cusparse_init(). See CUDA-specific Optimizations for more details.

57.5 Data Interfaces

Data management is done at a high-level in StarPU: rather than accessing a mere list of contiguous buffers, the tasks may manipulate data that are described by a high-level construct which we call data interface.

Data Structures

- struct starpu_data_copy_methods
- struct starpu_data_interface_ops
- struct starpu_matrix_interface
- struct starpu_coo_interface
- struct starpu_block_interface
- struct starpu_tensor_interface
- struct starpu_ndim_interface
- struct starpu_vector_interface
- struct starpu_variable_interface
- struct starpu_csr_interface
- struct starpu_bcsr_interface
- struct starpu_multiformat_data_interface_ops
- struct starpu_multiformat_interface

Enumerations

- enum starpu_data_interface_id {
STARPU_UNKNOWN_INTERFACE_ID , STARPU_MATRIX_INTERFACE_ID , STARPU_BLOCK_INTERFACE_ID , STARPU_VECTOR_INTERFACE_ID ,
STARPU_CSR_INTERFACE_ID , STARPU_BCSR_INTERFACE_ID , STARPU_VARIABLE_INTERFACE_ID , STARPU_VOID_INTERFACE_ID ,
STARPU_MULTIFORMAT_INTERFACE_ID , STARPU_COO_INTERFACE_ID , STARPU_TENSOR_INTERFACE_ID , STARPU_NDIM_INTERFACE_ID ,
STARPU_MAX_INTERFACE_ID }

Accessing Matrix Data Interfaces

- struct starpu_data_interface_ops **starpu_interface_matrix_ops**
- void starpu_matrix_data_register (starpu_data_handle_t ∗handle, int home_node, uintptr_t ptr, uint32_t ld, uint32_t nx, uint32_t ny, size_t elemsize)
- void starpu_matrix_data_register_allocsize (starpu_data_handle_t ∗handle, int home_node, uintptr_t ptr, uint32_t ld, uint32_t nx, uint32_t ny, size_t elemsize, size_t allocsize)
- void starpu_matrix_ptr_register (starpu_data_handle_t handle, unsigned node, uintptr_t ptr, uintptr_t dev_handle, size_t offset, uint32_t ld)
- uint32_t starpu_matrix_get_nx (starpu_data_handle_t handle)
- uint32_t starpu_matrix_get_ny (starpu_data_handle_t handle)
- uint32_t starpu_matrix_get_local_ld (starpu_data_handle_t handle)
- uintptr_t starpu_matrix_get_local_ptr (starpu_data_handle_t handle)
- size_t starpu_matrix_get_elemsize (starpu_data_handle_t handle)
- size_t starpu_matrix_get_allocsize (starpu_data_handle_t handle)
- #define STARPU_MATRIX_GET_PTR(interface)
- #define STARPU_MATRIX_GET_DEV_HANDLE(interface)
- #define STARPU_MATRIX_GET_OFFSET(interface)
- #define STARPU_MATRIX_GET_NX(interface)
- #define STARPU_MATRIX_GET_NY(interface)
- #define STARPU_MATRIX_GET_LD(interface)
- #define STARPU_MATRIX_GET_ELEMSIZE(interface)
- #define STARPU_MATRIX_GET_ALLOCSIZE(interface)
- #define STARPU_MATRIX_SET_NX(interface, newnx)
- #define STARPU_MATRIX_SET_NY(interface, newny)
- #define STARPU_MATRIX_SET_LD(interface, newld)

Accessing COO Data Interfaces

- struct starpu_data_interface_ops **starpu_interface_coo_ops**
- void starpu_coo_data_register (starpu_data_handle_t ∗handleptr, int home_node, uint32_t nx, uint32_t ny, uint32_t n_values, uint32_t ∗columns, uint32_t ∗rows, uintptr_t values, size_t elemsize)
- #define STARPU_COO_GET_COLUMNS(interface)
- #define STARPU_COO_GET_COLUMNS_DEV_HANDLE(interface)
- #define STARPU_COO_GET_ROWS(interface)
- #define STARPU_COO_GET_ROWS_DEV_HANDLE(interface)
- #define STARPU_COO_GET_VALUES(interface)
- #define STARPU_COO_GET_VALUES_DEV_HANDLE(interface)
- #define STARPU_COO_GET_OFFSET
- #define STARPU_COO_GET_NX(interface)
- #define STARPU_COO_GET_NY(interface)
- #define STARPU_COO_GET_NVALUES(interface)
- #define STARPU_COO_GET_ELEMSIZE(interface)

Block Data Interface

- struct starpu_data_interface_ops **starpu_interface_block_ops**
- void starpu_block_data_register (starpu_data_handle_t ∗handle, int home_node, uintptr_t ptr, uint32_t ldy, uint32_t ldz, uint32_t nx, uint32_t ny, uint32_t nz, size_t elemsize)
- void starpu_block_ptr_register (starpu_data_handle_t handle, unsigned node, uintptr_t ptr, uintptr_t dev_handle, size_t offset, uint32_t ldy, uint32_t ldz)
- uint32_t starpu_block_get_nx (starpu_data_handle_t handle)
- uint32_t starpu_block_get_ny (starpu_data_handle_t handle)
- uint32_t starpu_block_get_nz (starpu_data_handle_t handle)
- uint32_t starpu_block_get_local_ldy (starpu_data_handle_t handle)
- uint32_t starpu_block_get_local_ldz (starpu_data_handle_t handle)
- uintptr_t starpu_block_get_local_ptr (starpu_data_handle_t handle)
- size_t starpu_block_get_elemsize (starpu_data_handle_t handle)
- #define STARPU_BLOCK_GET_PTR(interface)
- #define STARPU_BLOCK_GET_DEV_HANDLE(interface)
- #define STARPU_BLOCK_GET_OFFSET(interface)
- #define STARPU_BLOCK_GET_NX(interface)
- #define STARPU_BLOCK_GET_NY(interface)
- #define STARPU_BLOCK_GET_NZ(interface)
- #define STARPU_BLOCK_GET_LDY(interface)
- #define STARPU_BLOCK_GET_LDZ(interface)
- #define STARPU_BLOCK_GET_ELEMSIZE(interface)

Tensor Data Interface

- struct starpu_data_interface_ops **starpu_interface_tensor_ops**
- void starpu_tensor_data_register (starpu_data_handle_t ∗handle, int home_node, uintptr_t ptr, uint32_t ldy, uint32_t ldz, uint32_t ldt, uint32_t nx, uint32_t ny, uint32_t nz, uint32_t nt, size_t elemsize)
- void starpu_tensor_ptr_register (starpu_data_handle_t handle, unsigned node, uintptr_t ptr, uintptr_t dev_handle, size_t offset, uint32_t ldy, uint32_t ldz, uint32_t ldt)
- uint32_t starpu_tensor_get_nx (starpu_data_handle_t handle)
- uint32_t starpu_tensor_get_ny (starpu_data_handle_t handle)
- uint32_t starpu_tensor_get_nz (starpu_data_handle_t handle)
- uint32_t starpu_tensor_get_nt (starpu_data_handle_t handle)
- uint32_t starpu_tensor_get_local_ldy (starpu_data_handle_t handle)
- uint32_t starpu_tensor_get_local_ldz (starpu_data_handle_t handle)
- uint32_t starpu_tensor_get_local_ldt (starpu_data_handle_t handle)
- uintptr_t starpu_tensor_get_local_ptr (starpu_data_handle_t handle)

- size_t starpu_tensor_get_elemsize (starpu_data_handle_t handle)
- #define STARPU_TENSOR_GET_PTR(interface)
- #define STARPU_TENSOR_GET_DEV_HANDLE(interface)
- #define STARPU_TENSOR_GET_OFFSET(interface)
- #define STARPU_TENSOR_GET_NX(interface)
- #define STARPU_TENSOR_GET_NY(interface)
- #define STARPU_TENSOR_GET_NZ(interface)
- #define STARPU_TENSOR_GET_NT(interface)
- #define STARPU_TENSOR_GET_LDY(interface)
- #define STARPU_TENSOR_GET_LDZ(interface)
- #define STARPU_TENSOR_GET_LDT(interface)
- #define STARPU_TENSOR_GET_ELEMSIZE(interface)

Ndim Array Data Interface

- struct starpu_data_interface_ops **starpu_interface_ndim_ops**
- void starpu_ndim_data_register (starpu_data_handle_t ∗handleptr, int home_node, uintptr_t ptr, uint32_t ∗ldn, uint32_t ∗nn, size_t ndim, size_t elemsize)
- void starpu_ndim_ptr_register (starpu_data_handle_t handle, unsigned node, uintptr_t ptr, uintptr_t dev_handle, size_t offset, uint32_t ∗ldn)
- uint32_t ∗ starpu_ndim_get_nn (starpu_data_handle_t handle)
- uint32_t starpu_ndim_get_ni (starpu_data_handle_t handle, size_t i)
- uint32_t ∗ starpu_ndim_get_local_ldn (starpu_data_handle_t handle)
- uint32_t starpu_ndim_get_local_ldi (starpu_data_handle_t handle, size_t i)
- uintptr_t starpu_ndim_get_local_ptr (starpu_data_handle_t handle)
- size_t starpu_ndim_get_ndim (starpu_data_handle_t handle)
- size_t starpu_ndim_get_elemsize (starpu_data_handle_t handle)
- #define STARPU_NDIM_GET_PTR(interface)
- #define STARPU_NDIM_GET_DEV_HANDLE(interface)
- #define STARPU_NDIM_GET_OFFSET(interface)
- #define STARPU_NDIM_GET_NN(interface)
- #define STARPU_NDIM_GET_LDN(interface)
- #define STARPU_NDIM_GET_NDIM(interface)
- #define STARPU_NDIM_GET_ELEMSIZE(interface)

Vector Data Interface

- struct starpu_data_interface_ops **starpu_interface_vector_ops**
- void starpu_vector_data_register (starpu_data_handle_t ∗handle, int home_node, uintptr_t ptr, uint32_t nx, size_t elemsize)
- void starpu_vector_data_register_allocsize (starpu_data_handle_t ∗handle, int home_node, uintptr_t ptr, uint32_t nx, size_t elemsize, size_t allocsize)
- void starpu_vector_ptr_register (starpu_data_handle_t handle, unsigned node, uintptr_t ptr, uintptr_t dev_handle, size_t offset)
- uint32_t starpu_vector_get_nx (starpu_data_handle_t handle)
- size_t starpu_vector_get_elemsize (starpu_data_handle_t handle)
- size_t starpu_vector_get_allocsize (starpu_data_handle_t handle)
- uintptr_t starpu_vector_get_local_ptr (starpu_data_handle_t handle)
- #define STARPU_VECTOR_GET_PTR(interface)
- #define STARPU_VECTOR_GET_DEV_HANDLE(interface)
- #define STARPU_VECTOR_GET_OFFSET(interface)
- #define STARPU_VECTOR_GET_NX(interface)
- #define STARPU_VECTOR_GET_ELEMSIZE(interface)
- #define STARPU_VECTOR_GET_ALLOCSIZE(interface)
- #define STARPU_VECTOR_GET_SLICE_BASE(interface)
- #define STARPU_VECTOR_SET_NX(interface, newnx)

Variable Data Interface

- struct starpu_data_interface_ops **starpu_interface_variable_ops**
- void starpu_variable_data_register (starpu_data_handle_t ∗handle, int home_node, uintptr_t ptr, size_t size)
- void starpu_variable_ptr_register (starpu_data_handle_t handle, unsigned node, uintptr_t ptr, uintptr_t dev_handle, size_t offset)
- size_t starpu_variable_get_elemsize (starpu_data_handle_t handle)
- uintptr_t starpu_variable_get_local_ptr (starpu_data_handle_t handle)
- #define STARPU_VARIABLE_GET_PTR(interface)
- #define STARPU_VARIABLE_GET_OFFSET(interface)
- #define STARPU_VARIABLE_GET_ELEMSIZE(interface)
- #define STARPU_VARIABLE_GET_DEV_HANDLE(interface)

Void Data Interface

- struct starpu_data_interface_ops **starpu_interface_void_ops**
- void starpu_void_data_register (starpu_data_handle_t ∗handle)

CSR Data Interface

- struct starpu_data_interface_ops **starpu_interface_csr_ops**
- void starpu_csr_data_register (starpu_data_handle_t ∗handle, int home_node, uint32_t nnz, uint32_t nrow, uintptr_t nzval, uint32_t ∗colind, uint32_t ∗rowptr, uint32_t firstentry, size_t elemsize)
- uint32_t starpu_csr_get_nnz (starpu_data_handle_t handle)
- uint32_t starpu_csr_get_nrow (starpu_data_handle_t handle)
- uint32_t starpu_csr_get_firstentry (starpu_data_handle_t handle)
- uintptr_t starpu_csr_get_local_nzval (starpu_data_handle_t handle)
- uint32_t ∗ starpu_csr_get_local_colind (starpu_data_handle_t handle)
- uint32_t ∗ starpu_csr_get_local_rowptr (starpu_data_handle_t handle)
- size_t starpu_csr_get_elemsize (starpu_data_handle_t handle)
- #define STARPU_CSR_GET_NNZ(interface)
- #define STARPU_CSR_GET_NROW(interface)
- #define STARPU_CSR_GET_NZVAL(interface)
- #define STARPU_CSR_GET_NZVAL_DEV_HANDLE(interface)
- #define STARPU_CSR_GET_COLIND(interface)
- #define STARPU_CSR_GET_RAM_COLIND(interface)
- #define STARPU_CSR_GET_COLIND_DEV_HANDLE(interface)
- #define STARPU_CSR_GET_ROWPTR(interface)
- #define STARPU_CSR_GET_RAM_ROWPTR(interface)
- #define STARPU_CSR_GET_ROWPTR_DEV_HANDLE(interface)
- #define STARPU_CSR_GET_OFFSET
- #define STARPU_CSR_GET_FIRSTENTRY(interface)
- #define STARPU_CSR_GET_ELEMSIZE(interface)

BCSR Data Interface

- struct starpu_data_interface_ops **starpu_interface_bcsr_ops**
- void starpu_bcsr_data_register (starpu_data_handle_t ∗handle, int home_node, uint32_t nnz, uint32_t nrow, uintptr_t nzval, uint32_t ∗colind, uint32_t ∗rowptr, uint32_t firstentry, uint32_t r, uint32_t c, size_t elemsize)
- uint32_t starpu_bcsr_get_nnz (starpu_data_handle_t handle)
- uint32_t starpu_bcsr_get_nrow (starpu_data_handle_t handle)
- uint32_t starpu_bcsr_get_firstentry (starpu_data_handle_t handle)
- uintptr_t starpu_bcsr_get_local_nzval (starpu_data_handle_t handle)
- uint32_t ∗ starpu_bcsr_get_local_colind (starpu_data_handle_t handle)
- uint32_t ∗ starpu_bcsr_get_local_rowptr (starpu_data_handle_t handle)
- uint32_t starpu_bcsr_get_r (starpu_data_handle_t handle)

- uint32_t starpu_bcsr_get_c (starpu_data_handle_t handle)
- size_t starpu_bcsr_get_elemsize (starpu_data_handle_t handle)
- #define STARPU_BCSR_GET_NNZ(interface)
- #define STARPU_BCSR_GET_NROW(interface)
- #define STARPU_BCSR_GET_NZVAL(interface)
- #define STARPU_BCSR_GET_NZVAL_DEV_HANDLE(interface)
- #define STARPU_BCSR_GET_COLIND(interface)
- #define STARPU_BCSR_GET_RAM_COLIND(interface)
- #define STARPU_BCSR_GET_COLIND_DEV_HANDLE(interface)
- #define STARPU_BCSR_GET_ROWPTR(interface)
- #define STARPU_BCSR_GET_RAM_ROWPTR(interface)
- #define STARPU_BCSR_GET_ROWPTR_DEV_HANDLE(interface)
- #define STARPU_BCSR_GET_FIRSTENTRY(interface)
- #define STARPU_BCSR_GET_R(interface)
- #define STARPU_BCSR_GET_C(interface)
- #define STARPU_BCSR_GET_ELEMSIZE(interface)
- #define STARPU_BCSR_GET_OFFSET

Basic API

- void starpu_data_register (starpu_data_handle_t ∗handleptr, int home_node, void ∗data_interface, struct starpu_data_interface_ops ∗ops)
- void starpu_data_register_ops (struct starpu_data_interface_ops ∗ops)
- void starpu_data_ptr_register (starpu_data_handle_t handle, unsigned node)
- void starpu_data_register_same (starpu_data_handle_t ∗handledst, starpu_data_handle_t handlesrc)
- void ∗ starpu_data_handle_to_pointer (starpu_data_handle_t handle, unsigned node)
- void ∗ starpu_data_get_local_ptr (starpu_data_handle_t handle)
- void ∗ starpu_data_get_interface_on_node (starpu_data_handle_t handle, unsigned memory_node)
- enum starpu_data_interface_id starpu_data_get_interface_id (starpu_data_handle_t handle)
- int starpu_data_pack_node (starpu_data_handle_t handle, unsigned node, void ∗∗ptr, starpu_ssize_t ∗count)
- int starpu_data_pack (starpu_data_handle_t handle, void ∗∗ptr, starpu_ssize_t ∗count)
- int starpu_data_peek_node (starpu_data_handle_t handle, unsigned node, void ∗ptr, size_t count)
- int starpu_data_peek (starpu_data_handle_t handle, void ∗ptr, size_t count)
- int starpu_data_unpack_node (starpu_data_handle_t handle, unsigned node, void ∗ptr, size_t count)
- int starpu_data_unpack (starpu_data_handle_t handle, void ∗ptr, size_t count)
- size_t starpu_data_get_size (starpu_data_handle_t handle)
- size_t starpu_data_get_alloc_size (starpu_data_handle_t handle)
- starpu_ssize_t starpu_data_get_max_size (starpu_data_handle_t handle)
- int starpu_data_get_home_node (starpu_data_handle_t handle)
- void starpu_data_print (starpu_data_handle_t handle, unsigned node, FILE ∗stream)
- int starpu_data_interface_get_next_id (void)
- int starpu_interface_copy (uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, size_t dst_offset, unsigned dst_node, size_t size, void ∗async_data)
- int starpu_interface_copy2d (uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, size_t dst_offset, unsigned dst_node, size_t blocksize, size_t numblocks, size_t ld_src, size_t ld_dst, void ∗async_data)
- int starpu_interface_copy3d (uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, size_t dst_offset, unsigned dst_node, size_t blocksize, size_t numblocks1, size_t ld1_src, size_t ld1_dst, size_t numblocks2, size_t ld2_src, size_t ld2_dst, void ∗async_data)
- int starpu_interface_copy4d (uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, size_t dst_offset, unsigned dst_node, size_t blocksize, size_t numblocks1, size_t ld1_src, size_t ld1_dst, size_t numblocks2, size_t ld2_src, size_t ld2_dst, size_t numblocks3, size_t ld3_src, size_t ld3_dst, void ∗async_data)
- int starpu_interface_copynd (uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, size_t dst_offset, unsigned dst_node, size_t elemsize, size_t ndim, uint32_t ∗nn, uint32_t ∗ldn_src, uint32_t ∗ldn_dst, void ∗async_data)
- void starpu_interface_start_driver_copy_async (unsigned src_node, unsigned dst_node, double ∗start)

- void starpu_interface_end_driver_copy_async (unsigned src_node, unsigned dst_node, double start)
- void starpu_interface_data_copy (unsigned src_node, unsigned dst_node, size_t size)
- uintptr_t starpu_malloc_on_node_flags (unsigned dst_node, size_t size, int flags)
- uintptr_t starpu_malloc_on_node (unsigned dst_node, size_t size)
- void starpu_free_on_node_flags (unsigned dst_node, uintptr_t addr, size_t size, int flags)
- void starpu_free_on_node (unsigned dst_node, uintptr_t addr, size_t size)
- void starpu_malloc_on_node_set_default_flags (unsigned node, int flags)

MAP API

- uintptr_t starpu_interface_map (uintptr_t src, size_t src_offset, unsigned src_node, unsigned dst_node, size_t size, int ∗ret)
- int starpu_interface_unmap (uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, unsigned dst_node, size_t size)
- int starpu_interface_update_map (uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, size_t dst_offset, unsigned dst_node, size_t size)

Multiformat Data Interface

- void starpu_multiformat_data_register (starpu_data_handle_t ∗handle, int home_node, void ∗ptr, uint32_t nobjects, struct starpu_multiformat_data_interface_ops ∗format_ops)
- #define STARPU_MULTIFORMAT_GET_CPU_PTR(interface)
- #define STARPU_MULTIFORMAT_GET_CUDA_PTR(interface)
- #define STARPU_MULTIFORMAT_GET_HIP_PTR(interface)
- #define STARPU_MULTIFORMAT_GET_OPENCL_PTR(interface)
- #define STARPU_MULTIFORMAT_GET_NX(interface)

- uint32_t starpu_hash_crc32c_be_n (const void ∗input, size_t n, uint32_t inputcrc)
- uint32_t starpu_hash_crc32c_be_ptr (void ∗input, uint32_t inputcrc)
- uint32_t starpu_hash_crc32c_be (uint32_t input, uint32_t inputcrc)
- uint32_t starpu_hash_crc32c_string (const char ∗str, uint32_t inputcrc)

57.5.1 Detailed Description

Data management is done at a high-level in StarPU: rather than accessing a mere list of contiguous buffers, the tasks may manipulate data that are described by a high-level construct which we call data interface.

An example of data interface is the "vector" interface which describes a contiguous data array on a specific memory node. This interface is a simple structure containing the number of elements in the array, the size of the elements, and the address of the array in the appropriate address space (this address may be invalid if there is no valid copy of the array in the memory node). More information on the data interfaces provided by StarPU are given in Data Interfaces.

When a piece of data managed by StarPU is used by a task, the task implementation is given a pointer to an interface describing a valid copy of the data that is accessible from the current processing unit.

Every worker is associated to a memory node which is a logical abstraction of the address space from which the processing unit gets its data. For instance, the memory node associated to the different CPU workers represents main memory (RAM), the memory node associated to a GPU is DRAM embedded on the device. Every memory node is identified by a logical index which is accessible from the function starpu_worker_get_memory_node(). When registering a piece of data to StarPU, the specified memory node indicates where the piece of data initially resides (we also call this memory node the home node of a piece of data).

In the case of NUMA systems, functions starpu_memory_nodes_numa_devid_to_id() and starpu_memory_nodes_numa_id_to_devid(can be used to convert from NUMA node numbers as seen by the Operating System and NUMA node numbers as seen by StarPU.

There are several ways to register a memory region so that it can be managed by StarPU. StarPU provides data interfaces for vectors, 2D matrices, 3D matrices as well as BCSR and CSR sparse matrices.

Each data interface is provided with a set of field access functions. The ones using a `void ∗` parameter aimed to be used in codelet implementations (see for example the code in Vector Scaling).

Applications can provide their own interface as shown in Defining A New Data Interface.

57.5.2 Data Structure Documentation

57.5.2.1 struct starpu_data_copy_methods

Define the per-interface methods. If the starpu_data_copy_methods::any_to_any method is provided, it will be used by default if no specific method is provided. It can still be useful to provide more specific method in case of e.g. available particular CUDA, HIP or OpenCL support.
See Data copy for more details.

Data Fields

- int(∗ can_copy)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node, unsigned handling_node)
- int(∗ ram_to_ram)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node)
- int(∗ ram_to_cuda)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node)
- int(∗ ram_to_hip)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node)
- int(∗ ram_to_opencl)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node)
- int(∗ ram_to_max_fpga)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node)
- int(∗ cuda_to_ram)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node)
- int(∗ cuda_to_cuda)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node)
- int(∗ hip_to_ram)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node)
- int(∗ hip_to_hip)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node)
- int(∗ opencl_to_ram)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node)
- int(∗ opencl_to_opencl)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node)
- int(∗ max_fpga_to_ram)(void ∗src_interface, unsigned srd_node, void ∗dst_interface, unsigned dst_node)
- int(∗ ram_to_cuda_async)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node, starpu_cudaStream_t stream)
- int(∗ cuda_to_ram_async)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node, starpu_cudaStream_t stream)
- int(∗ cuda_to_cuda_async)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node, starpu_cudaStream_t stream)
- int(∗ ram_to_hip_async)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node, starpu_hipStream_t stream)
- int(∗ hip_to_ram_async)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node, starpu_hipStream_t stream)
- int(∗ hip_to_hip_async)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node, starpu_hipStream_t stream)
- int(∗ ram_to_opencl_async)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node, cl_event ∗event)
- int(∗ opencl_to_ram_async)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node, cl_event ∗event)
- int(∗ opencl_to_opencl_async)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node, cl_event ∗event)
- int(∗ ram_to_max_fpga_async)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node)
- int(∗ max_fpga_to_ram_async)(void ∗src_interface, unsigned srd_node, void ∗dst_interface, unsigned dst_node)
- int(∗ any_to_any)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node, void ∗async_data)

57.5.2.1.1 Field Documentation

57.5.2.1.1.1 can_copy `int(∗ starpu_data_copy_methods::can_copy) (void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node, unsigned handling_node)`
If defined, allow the interface to declare whether it supports transferring from `src_interface` on node `src_node` to `dst_interface` on node `dst_node`, run from node `handling_node`. If not defined, it is assumed that the interface supports all transfers.

57.5.2.1.1.2 ram_to_ram `int(* starpu_data_copy_methods::ram_to_ram) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` CPU node to the `dst_interface` interface on the `dst_node` CPU node. Return 0 on success.

57.5.2.1.1.3 ram_to_cuda `int(* starpu_data_copy_methods::ram_to_cuda) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` CPU node to the `dst_interface` interface on the `dst_node` CUDA node. Return 0 on success.

57.5.2.1.1.4 ram_to_hip `int(* starpu_data_copy_methods::ram_to_hip) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` CPU node to the `dst_interface` interface on the `dst_node` HIP node. Return 0 on success.

57.5.2.1.1.5 ram_to_opencl `int(* starpu_data_copy_methods::ram_to_opencl) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` CPU node to the `dst_interface` interface on the `dst_node` OpenCL node. Return 0 on success.

57.5.2.1.1.6 ram_to_max_fpga `int(* starpu_data_copy_methods::ram_to_max_fpga) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` CPU node to the `dst_interface` interface on the `dst_node` FPGA node. Return 0 on success.

57.5.2.1.1.7 cuda_to_ram `int(* starpu_data_copy_methods::cuda_to_ram) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` CUDA node to the `dst_interface` interface on the `dst_node` CPU node. Return 0 on success.

57.5.2.1.1.8 cuda_to_cuda `int(* starpu_data_copy_methods::cuda_to_cuda) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` CUDA node to the `dst_interface` interface on the `dst_node` CUDA node. Return 0 on success.

57.5.2.1.1.9 hip_to_ram `int(* starpu_data_copy_methods::hip_to_ram) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` HIP node to the `dst_interface` interface on the `dst_node` CPU node. Return 0 on success.

57.5.2.1.1.10 hip_to_hip `int(* starpu_data_copy_methods::hip_to_hip) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` HIP node to the `dst_interface` interface on the `dst_node` HIP node. Return 0 on success.

57.5.2.1.1.11 opencl_to_ram `int(* starpu_data_copy_methods::opencl_to_ram) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` OpenCL node to the `dst_interface` interface on the `dst_node` CPU node. Return 0 on success.

57.5.2.1.1.12 opencl_to_opencl `int(* starpu_data_copy_methods::opencl_to_opencl) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` OpenCL node to the `dst_interface` interface on the `dst_node` OpenCL node. Return 0 on success.

57.5.2.1.1.13 max_fpga_to_ram `int(* starpu_data_copy_methods::max_fpga_to_ram) (void *src_interface, unsigned srd_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` FPGA node to the `dst_interface` interface on the `dst_node` CPU node. Return 0 on success.

57.5.2.1.1.14 ram_to_cuda_async `int(* starpu_data_copy_methods::ram_to_cuda_async) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, starpu_cudaStream_t stream)`

Define how to copy data from the `src_interface` interface on the `src_node` CPU node to the `dst_interface` interface on the `dst_node` CUDA node, using the given stream. Must return 0 if the transfer was actually completed completely synchronously, or `-EAGAIN` if at least some transfers are still ongoing and should be awaited for by the core.

57.5.2.1.1.15 cuda_to_ram_async `int(* starpu_data_copy_methods::cuda_to_ram_async) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, starpu_cudaStream_t stream)`

Define how to copy data from the `src_interface` interface on the `src_node` CUDA node to the `dst_interface` interface on the `dst_node` CPU node, using the given stream. Must return 0 if the transfer was actually completed completely synchronously, or `-EAGAIN` if at least some transfers are still ongoing and should be awaited for by the core.

57.5.2.1.1.16 cuda_to_cuda_async `int(* starpu_data_copy_methods::cuda_to_cuda_async) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, starpu_cudaStream_t stream)`

Define how to copy data from the `src_interface` interface on the `src_node` CUDA node to the `dst_interface` interface on the `dst_node` CUDA node, using the given stream. Must return 0 if the transfer was actually completed completely synchronously, or `-EAGAIN` if at least some transfers are still ongoing and should be awaited for by the core.

57.5.2.1.1.17 ram_to_hip_async `int(* starpu_data_copy_methods::ram_to_hip_async) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, starpu_hipStream_t stream)`

Define how to copy data from the `src_interface` interface on the `src_node` CPU node to the `dst_interface` interface on the `dst_node` HIP node, using the given stream. Must return 0 if the transfer was actually completed completely synchronously, or `-EAGAIN` if at least some transfers are still ongoing and should be awaited for by the core.

57.5.2.1.1.18 hip_to_ram_async `int(* starpu_data_copy_methods::hip_to_ram_async) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, starpu_hipStream_t stream)`

Define how to copy data from the `src_interface` interface on the `src_node` HIP node to the `dst_interface` interface on the `dst_node` CPU node, using the given stream. Must return 0 if the transfer was actually completed completely synchronously, or `-EAGAIN` if at least some transfers are still ongoing and should be awaited for by the core.

57.5.2.1.1.19 hip_to_hip_async `int(* starpu_data_copy_methods::hip_to_hip_async) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, starpu_hipStream_t stream)`

Define how to copy data from the `src_interface` interface on the `src_node` HIP node to the `dst_interface` interface on the `dst_node` HIP node, using the given stream. Must return 0 if the transfer was actually completed completely synchronously, or `-EAGAIN` if at least some transfers are still ongoing and should be awaited for by the core.

57.5.2.1.1.20 ram_to_opencl_async `int(* starpu_data_copy_methods::ram_to_opencl_async) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, cl_event *event)`

Define how to copy data from the `src_interface` interface on the `src_node` CPU node to the `dst_interface` interface on the `dst_node` OpenCL node, by recording in `event`, a pointer to a `cl_event`, the event of the last submitted transfer. Must return 0 if the transfer was actually completed completely synchronously, or `-EAGAIN` if at least some transfers are still ongoing and should be awaited for by the core.

57.5.2.1.1.21 opencl_to_ram_async `int(* starpu_data_copy_methods::opencl_to_ram_async) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, cl_event *event)`

Define how to copy data from the `src_interface` interface on the `src_node` OpenCL node to the `dst_interface` interface on the `dst_node` CPU node, by recording in `event`, a pointer to a `cl_event`, the event of the last submitted transfer. Must return 0 if the transfer was actually completed completely synchronously, or `-EAGAIN` if at least some transfers are still ongoing and should be awaited for by the core.

57.5.2.1.1.22 opencl_to_opencl_async `int(* starpu_data_copy_methods::opencl_to_opencl_async) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, cl_event *event)`

Define how to copy data from the `src_interface` interface on the `src_node` OpenCL node to the `dst_interface` interface on the `dst_node` OpenCL node, by recording in `event`, a pointer to a `cl_event`, the event of the last submitted transfer. Must return 0 if the transfer was actually completed completely synchronously, or `-EAGAIN` if at least some transfers are still ongoing and should be awaited for by the core.

57.5.2.1.1.23 ram_to_max_fpga_async `int(* starpu_data_copy_methods::ram_to_max_fpga_async) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` CPU node to the `dst_interface` interface on the `dst_node` FPGA node. Must return 0 if the transfer was actually completed completely synchronously, or `-EAGAIN` if at least some transfers are still ongoing and should be awaited for by the core.

57.5.2.1.1.24 max_fpga_to_ram_async `int(* starpu_data_copy_methods::max_fpga_to_ram_async) (void *src_interface, unsigned srd_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` FPGA node to the `dst_interface` interface on the `dst_node` CPU node. Must return 0 if the transfer was actually completed completely synchronously, or `-EAGAIN` if at least some transfers are still ongoing and should be awaited for by the core.

57.5.2.1.1.25 any_to_any `int(* starpu_data_copy_methods::any_to_any) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, void *async_data)`

Define how to copy data from the `src_interface` interface on the `src_node` node to the `dst_interface` interface on the `dst_node` node. This is meant to be implemented through the starpu_interface_copy() helper, to which async_data should be passed as such, and will be used to manage asynchronicity. This must return `-EAGAIN` if any of the starpu_interface_copy() calls has returned `-EAGAIN` (i.e. at least some transfer is still ongoing), and return 0 otherwise.
This can only be implemented if the interface has ready-to-send data blocks. If the interface is more involved than this, i.e. it needs to collect pieces of data before transferring, starpu_data_interface_ops::pack_data and starpu_data_interface_ops::peek_data should be implemented instead, and the core will just transfer the resulting data buffer.

57.5.2.2 struct starpu_data_interface_ops

Per-interface data management methods.

Data Fields

- void(∗ register_data_handle)(starpu_data_handle_t handle, int home_node, void ∗data_interface)
- void(∗ unregister_data_handle)(starpu_data_handle_t handle)
- starpu_ssize_t(∗ allocate_data_on_node)(void ∗data_interface, unsigned node)
- void(∗ free_data_on_node)(void ∗data_interface, unsigned node)
- void(∗ cache_data_on_node)(void ∗cached_interface, void ∗src_interface, unsigned node)
- void(∗ reuse_data_on_node)(void ∗dst_data_interface, const void ∗cached_interface, unsigned node)
- int(∗ map_data)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node)

- int(∗ unmap_data)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node)
- int(∗ update_map)(void ∗src_interface, unsigned src_node, void ∗dst_interface, unsigned dst_node)
- void(∗ init)(void ∗data_interface)
- const struct starpu_data_copy_methods ∗ copy_methods
- void ∗(∗ handle_to_pointer)(starpu_data_handle_t handle, unsigned node)
- void ∗(∗ to_pointer)(void ∗data_interface, unsigned node)
- size_t(∗ get_size)(starpu_data_handle_t handle)
- size_t(∗ get_alloc_size)(starpu_data_handle_t handle)
- size_t(∗ get_max_size)(starpu_data_handle_t handle)
- uint32_t(∗ footprint)(starpu_data_handle_t handle)
- uint32_t(∗ alloc_footprint)(starpu_data_handle_t handle)
- int(∗ compare)(void ∗data_interface_a, void ∗data_interface_b)
- int(∗ alloc_compare)(void ∗data_interface_a, void ∗data_interface_b)
- void(∗ display)(starpu_data_handle_t handle, FILE ∗f)
- starpu_ssize_t(∗ describe)(void ∗data_interface, char ∗buf, size_t size)
- enum starpu_data_interface_id interfaceid
- size_t interface_size
- char **is_multiformat**
- char dontcache
- struct starpu_multiformat_data_interface_ops ∗(∗ **get_mf_ops**)(void ∗data_interface)
- int(∗ pack_data)(starpu_data_handle_t handle, unsigned node, void ∗∗ptr, starpu_ssize_t ∗count)
- int(∗ peek_data)(starpu_data_handle_t handle, unsigned node, void ∗ptr, size_t count)
- int(∗ unpack_data)(starpu_data_handle_t handle, unsigned node, void ∗ptr, size_t count)
- int(∗ pack_meta)(void ∗data_interface, void ∗∗ptr, starpu_ssize_t ∗count)
- int(∗ unpack_meta)(void ∗∗data_interface, void ∗ptr, starpu_ssize_t ∗count)
- int(∗ free_meta)(void ∗data_interface)
- char ∗ name

57.5.2.2.1 Field Documentation

57.5.2.2.1.1 register_data_handle `void(∗ starpu_data_interface_ops::register_data_handle) (starpu_data_handle_t handle, int home_node, void ∗data_interface)`

Register an existing interface into a data handle.
This iterates over all memory nodes to initialize all fields of the data interface on each of them. Since data is not allocated yet except on the home node, pointers should be left as NULL except on the `home_node` (if >= 0), for which the pointers should be copied from the given `data_interface`, which was filled with the application's pointers.
This method is mandatory.
See Data registration for more details.

57.5.2.2.1.2 unregister_data_handle `void(∗ starpu_data_interface_ops::unregister_data_handle) (starpu_data_handle_t handle)`

Unregister a data handle.
This iterates over all memory nodes to free any pointer in the data interface on each of them.
At this point, free_data_on_node has been already called on each of them. This just clears anything that would still be left.
See Data registration for more details.

57.5.2.2.1.3 allocate_data_on_node `starpu_ssize_t(∗ starpu_data_interface_ops::allocate_data_on_node) (void ∗data_interface, unsigned node)`

Allocate data for the interface on a given node. This should use starpu_malloc_on_node() to perform the allocation(s), and fill the pointers in the data interface. It should return the size of the allocated memory, or -ENOMEM if memory could not be allocated.
Note that the memory node can be CPU memory, GPU memory, or even disk area. The result returned by starpu_malloc_on_node() should be just stored as uintptr_t without trying to interpret it since it may be a GPU pointer, a disk descriptor, etc.

This method is mandatory to be able to support memory nodes.
See Pointers inside the data interface for more details.

57.5.2.2.1.4 free_data_on_node `void(* starpu_data_interface_ops::free_data_on_node) (void *data_interface, unsigned node)`

Free data of the interface on a given node.
This method is mandatory to be able to support memory nodes.
See Pointers inside the data interface for more details.

57.5.2.2.1.5 cache_data_on_node `void(* starpu_data_interface_ops::cache_data_on_node) (void *cached_interface, void *src_interface, unsigned node)`

Cache the buffers from the given node to a caching interface.
This method is optional, mostly useful when also making starpu_data_interface_ops::unregister_data_handle check that pointers are NULL.
`src_interface` is an interface that already has buffers allocated, but which we don't need any more. `cached_interface` is a new interface into which the buffer pointers should be transferred, for later reuse when allocating data of the same kind.
Usually we can just memcpy over the set of pointers and descriptions (this is what StarPU does when this method is not implemented), but if unregister_data_handle checks that pointers are NULL, we need to additionally clear the pointers in `src_interface`. Also, it is not useful to copy the whole interface, only the pointers need to be copied (essentially the pointers that starpu_data_interface_ops::reuse_data_on_node will then transfer into a new handle interface), as well as the properties that starpu_data_interface_ops::compare (or starpu_data_interface_ops::alloc_compare if defined) needs for comparing interfaces for caching compatibility.
When this method is not defined, StarPU will just copy the `cached_interface` into `src_interface`.
See Data Interface with Variable Size and Pointers inside the data interface for more details.

57.5.2.2.1.6 reuse_data_on_node `void(* starpu_data_interface_ops::reuse_data_on_node) (void *dst_data_interface, const void *cached_interface, unsigned node)`

Reuse on the given node the buffers of the provided interface
This method is optional, mostly useful when also defining alloc_footprint to share tiles of the same allocation size but different shapes, or when the interface contains pointers which are initialized at registration (e.g. nn array in the ndim interface)
`cached_interface` is an already-allocated buffer that we want to reuse, and `new_data_interface` is an interface in which we want to install that already-allocated buffer. Usually we can just memcpy over the set of pointers and descriptions. But e.g. with 2D tiles the ld value may not be correct, and memcpy would wrongly overwrite it in new_data_interface, i.e. reusing a vertical tile allocation for a horizontal tile, or vice-versa.
reuse_data_on_node should thus copy over pointers, and define fields that are usually set by allocate_data_on_node (e.g. ld).
See Data Interface with Variable Size and Pointers inside the data interface for more details.

57.5.2.2.1.7 map_data `int(* starpu_data_interface_ops::map_data) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Map data from a source to a destination. Define function starpu_interface_map() to set this field. See Pointers inside the data interface for more details.

57.5.2.2.1.8 unmap_data `int(* starpu_data_interface_ops::unmap_data) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Unmap data from a source to a destination. Define function starpu_interface_unmap() to set this field. See Pointers inside the data interface for more details.

57.5.2.2.1.9 update_map `int(* starpu_data_interface_ops::update_map) (void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Update map data from a source to a destination. Define function starpu_interface_update_map() to set this field.
See Pointers inside the data interface for more details.

57.5.2.2.1.10 init `void(* starpu_data_interface_ops::init) (void *data_interface)`

Initialize the interface. This method is optional. It is called when initializing the handler on all the memory nodes.

57.5.2.2.1.11 copy_methods `const struct starpu_data_copy_methods* starpu_data_interface_ops↩ ::copy_methods`

Struct with pointer to functions for performing ram/cuda/opencl synchronous and asynchronous transfers.
This field is mandatory to be able to support memory nodes, except disk nodes which can be supported by just implementing starpu_data_interface_ops::pack_data and starpu_data_interface_ops::unpack_data.

57.5.2.2.1.12 handle_to_pointer `void *(* starpu_data_interface_ops::handle_to_pointer) (starpu_data_handle_t handle, unsigned node)`

Deprecated Use starpu_data_interface_ops::to_pointer instead. Return the current pointer (if any) for the handle on the given node.

This method is only required if starpu_data_interface_ops::to_pointer is not implemented.

57.5.2.2.1.13 to_pointer `void *(* starpu_data_interface_ops::to_pointer) (void *data_interface, unsigned node)`

Return the current pointer (if any) for the given interface on the given node.
This method is only required for starpu_data_handle_to_pointer() and starpu_data_get_local_ptr(), and for disk support.

57.5.2.2.1.14 get_size `size_t(* starpu_data_interface_ops::get_size) (starpu_data_handle_t handle)`

Return an estimation of the size of data, for performance models and tracing feedback.

57.5.2.2.1.15 get_alloc_size `size_t(* starpu_data_interface_ops::get_alloc_size) (starpu_data_handle_t handle)`

Return an estimation of the size of allocated data, for allocation management. If not specified, the starpu_data_interface_ops::get_size method is used instead.

57.5.2.2.1.16 get_max_size `size_t(* starpu_data_interface_ops::get_max_size) (starpu_data_handle_t handle)`

Return the maximum size that the data may need to increase to. For instance, in the case of compressed matrix tiles this is the size when the block is fully dense. This is currently only used for feedback tools.

57.5.2.2.1.17 footprint `uint32_t(* starpu_data_interface_ops::footprint) (starpu_data_handle_t handle)`

Return a 32bit footprint which characterizes the data size and layout (nx, ny, ld, elemsize, etc.), required for indexing performance models.
starpu_hash_crc32c_be() and alike can be used to produce this 32bit value from various types of values.

57.5.2.2.1.18 alloc_footprint `uint32_t(* starpu_data_interface_ops::alloc_footprint) (starpu_data_handle_t handle)`

Return a 32bit footprint which characterizes the data allocation, to be used for indexing allocation cache. If not specified, the starpu_data_interface_ops::footprint method is used instead. If specified, alloc_compare should be set to provide the strict comparison, and reuse_data_on_node should be set to provide correct buffer reuse.

57.5.2.2.1.19 compare `int(* starpu_data_interface_ops::compare) (void *data_interface_a, void *data_interface_b)`

Compare the data size and layout of two interfaces (nx, ny, ld, elemsize, etc.), to be used for indexing performance models. It should return 1 if the two interfaces size and layout match computation-wise, and 0 otherwise. It does *not* compare the actual content of the interfaces.

57.5.2.2.1.20 alloc_compare `int(* starpu_data_interface_ops::alloc_compare) (void *data_↩ interface_a, void *data_interface_b)`

Compare the data allocation of two interfaces etc.), to be used for indexing allocation cache. It should return 1 if the two interfaces are allocation-compatible, i.e. basically have the same alloc_size, and 0 otherwise. If not specified, the starpu_data_interface_ops::compare method is used instead.

57.5.2.2.1.21 display `void(* starpu_data_interface_ops::display) (starpu_data_handle_t handle, FILE *f)`

Dump the sizes of a handle to a file. This is required for performance models

57.5.2.2.1.22 describe `starpu_ssize_t(* starpu_data_interface_ops::describe) (void *data_interface, char *buf, size_t size)`

Describe the data into a string in a brief way, such as one letter to describe the type of data, and the data dimensions. This is required for tracing feedback.

57.5.2.2.1.23 interfaceid `enum starpu_data_interface_id starpu_data_interface_ops::interfaceid`

An identifier that is unique to each interface.

57.5.2.2.1.24 interface_size `size_t starpu_data_interface_ops::interface_size`

Size of the interface data descriptor.

57.5.2.2.1.25 dontcache `char starpu_data_interface_ops::dontcache`

If set to non-zero, StarPU will never try to reuse an allocated buffer for a different handle. This can be notably useful for application-defined interfaces which have a dynamic size, and for which it thus does not make sense to reuse the buffer since will probably not have the proper size.

57.5.2.2.1.26 pack_data `int(* starpu_data_interface_ops::pack_data) (starpu_data_handle_t handle, unsigned node, void **ptr, starpu_ssize_t *count)`

Pack the data handle into a contiguous buffer at the address allocated with `starpu_malloc_flags(ptr, size, 0)` (and thus returned in `ptr`) and set the size of the newly created buffer in `count`. If `ptr` is `NULL`, the function should not copy the data in the buffer but just set count to the size of the buffer which would have been allocated. The special value -1 indicates the size is yet unknown.
This method (and starpu_data_interface_ops::unpack_data) is required for disk support if the starpu_data_copy_methods::any_to_any method is not implemented (because the in-memory data layout is too complex).
This is also required for MPI support if there is no registered MPI data type.

57.5.2.2.1.27 peek_data `int(* starpu_data_interface_ops::peek_data) (starpu_data_handle_t handle, unsigned node, void *ptr, size_t count)`

Read the data handle from the contiguous buffer at the address `ptr` of size `count`.

57.5.2.2.1.28 unpack_data `int(* starpu_data_interface_ops::unpack_data) (starpu_data_handle_t handle, unsigned node, void *ptr, size_t count)`

Unpack the data handle from the contiguous buffer at the address `ptr` of size `count`. The memory at the address `ptr` should be freed after the data unpacking operation.

57.5.2.2.1.29 pack_meta `int(* starpu_data_interface_ops::pack_meta) (void *data_interface, void **ptr, starpu_ssize_t *count)`

Pack the interface into a contiguous buffer and set the size of the newly created buffer in `count`. This function is used in master slave mode for data interfaces with a dynamic content.

57.5.2.2.1.30 unpack_meta `int(* starpu_data_interface_ops::unpack_meta) (void **data_interface, void *ptr, starpu_ssize_t *count)`

Unpack the interface from the given buffer and set the size of the unpacked data in `count`. This function is used in master slave mode for data interfaces with a dynamic content.

57.5.2.2.1.31 free_meta `int(* starpu_data_interface_ops::free_meta) (void *data_interface)`

Free the allocated memory by a previous call to unpack_meta()

57.5.2.2.1.32 name `char* starpu_data_interface_ops::name`

Name of the interface

57.5.2.3 struct starpu_matrix_interface

Matrix interface for dense matrices

Data Fields

enum starpu_data_interface_id	id	Identifier of the interface
uintptr_t	ptr	local pointer of the matrix
uintptr_t	dev_handle	device handle of the matrix
size_t	offset	offset in the matrix
uint32_t	nx	number of elements on the x-axis of the matrix
uint32_t	ny	number of elements on the y-axis of the matrix
uint32_t	ld	number of elements between each row of the matrix. Maybe be equal to starpu_matrix_interface::nx when there is no padding.
size_t	elemsize	size of the elements of the matrix
size_t	allocsize	size actually currently allocated

57.5.2.4 struct starpu_coo_interface

COO Matrices

Data Fields

enum starpu_data_interface_id	id	identifier of the interface
uint32_t ∗	columns	column array of the matrix
uint32_t ∗	rows	row array of the matrix
uintptr_t	values	values of the matrix
uint32_t	nx	number of elements on the x-axis of the matrix
uint32_t	ny	number of elements on the y-axis of the matrix
uint32_t	n_values	number of values registered in the matrix
size_t	elemsize	size of the elements of the matrix

57.5.2.5 struct starpu_block_interface

Block interface for 3D dense blocks

Data Fields

enum starpu_data_interface_id	id	identifier of the interface
uintptr_t	ptr	local pointer of the block
uintptr_t	dev_handle	device handle of the block.
size_t	offset	offset in the block.
uint32_t	nx	number of elements on the x-axis of the block.
uint32_t	ny	number of elements on the y-axis of the block.
uint32_t	nz	number of elements on the z-axis of the block.
uint32_t	ldy	number of elements between two lines
uint32_t	ldz	number of elements between two planes
size_t	elemsize	size of the elements of the block.

57.5.2.6 struct starpu_tensor_interface

Tensor interface for 4D dense tensors

Data Fields

enum starpu_data_interface_id	id	identifier of the interface
uintptr_t	ptr	local pointer of the tensor
uintptr_t	dev_handle	device handle of the tensor.
size_t	offset	offset in the tensor.
uint32_t	nx	number of elements on the x-axis of the tensor.
uint32_t	ny	number of elements on the y-axis of the tensor.
uint32_t	nz	number of elements on the z-axis of the tensor.
uint32_t	nt	number of elements on the t-axis of the tensor.
uint32_t	ldy	number of elements between two lines
uint32_t	ldz	number of elements between two planes
uint32_t	ldt	number of elements between two cubes
size_t	elemsize	size of the elements of the tensor.

57.5.2.7 struct starpu_ndim_interface

ndim interface for ndim array

Data Fields

enum starpu_data_interface_id	id	identifier of the interface
uintptr_t	ptr	local pointer of the ndim
uintptr_t	dev_handle	device handle of the ndim.
size_t	offset	offset in the ndim.
size_t	allocsize	size actually currently allocated.
uint32_t *	nn	array of element number on each dimension
uint32_t *	ldn	array of element number between two units on each dimension
size_t	ndim	size of the dimension.
size_t	elemsize	size of the elements of the ndim.

57.5.2.8 struct starpu_vector_interface

todo

Data Fields

enum starpu_data_interface_id	id	Identifier of the interface
uintptr_t	ptr	local pointer of the vector
uintptr_t	dev_handle	device handle of the vector.
size_t	offset	offset in the vector
uint32_t	nx	number of elements on the x-axis of the vector
size_t	elemsize	size of the elements of the vector
uint32_t	slice_base	vector slice base, used by the StarPU OpenMP runtime support
size_t	allocsize	size actually currently allocated

57.5.2.9 struct starpu_variable_interface

Variable interface for a single data (not a vector, a matrix, a list, ...)

Data Fields

enum starpu_data_interface_id	id	Identifier of the interface
uintptr_t	ptr	local pointer of the variable
uintptr_t	dev_handle	device handle of the variable.
size_t	offset	offset in the variable
size_t	elemsize	size of the variable

57.5.2.10 struct starpu_csr_interface

CSR interface for sparse matrices (compressed sparse row representation)

Data Fields

enum starpu_data_interface_id	id	Identifier of the interface
uint32_t	nnz	number of non-zero entries
uint32_t	nrow	number of rows
uintptr_t	nzval	non-zero values
uint32_t *	colind	position of non-zero entries on the row
uint32_t *	rowptr	index (in nzval) of the first entry of the row
uint32_t *	ram_colind	position of non-zero entries on the row (stored in RAM)
uint32_t *	ram_rowptr	index (in nzval) of the first entry of the row (stored in RAM)
uint32_t	firstentry	k for k-based indexing (0 or 1 usually). also useful when partitioning the matrix.
size_t	elemsize	size of the elements of the matrix

57.5.2.11 struct starpu_bcsr_interface

BCSR interface for sparse matrices (blocked compressed sparse row representation)
Note: when a BCSR matrix is partitioned, nzval, colind, and rowptr point into the corresponding father arrays. The rowptr content is thus the same as the father's. Firstentry is used to offset this so it becomes valid for the child arrays.

Data Fields

enum starpu_data_interface_id	id	Identifier of the interface
uint32_t	nnz	number of non-zero BLOCKS
uint32_t	nrow	number of rows (in terms of BLOCKS)
uintptr_t	nzval	non-zero values: nnz blocks of r*c elements
uint32_t *	colind	array of nnz elements, colind[i] is the block-column index for block i in nzval
uint32_t *	rowptr	array of nrow+1 elements, rowptr[i] is the block-index (in nzval) of the first block of row i. By convention, rowptr[nrow] is the number of blocks, this allows an easier access of the matrix's elements for the kernels.
uint32_t *	ram_colind	array of nnz elements (stored in RAM)
uint32_t *	ram_rowptr	array of nrow+1 elements (stored in RAM)
uint32_t	firstentry	k for k-based indexing (0 or 1 usually). Also useful when partitioning the matrix.

Data Fields

uint32_t	r	height of the blocks
uint32_t	c	width of the blocks
size_t	elemsize	size of the elements of the matrix

57.5.2.12 struct starpu_multiformat_data_interface_ops

Multiformat operations

Data Fields

size_t	cpu_elemsize	size of each element on CPUs
size_t	opencl_elemsize	size of each element on OpenCL devices
struct starpu_codelet *	cpu_to_opencl_cl	pointer to a codelet which converts from CPU to OpenCL
struct starpu_codelet *	opencl_to_cpu_cl	pointer to a codelet which converts from OpenCL to CPU
size_t	cuda_elemsize	size of each element on CUDA devices
struct starpu_codelet *	cpu_to_cuda_cl	pointer to a codelet which converts from CPU to CUDA
struct starpu_codelet *	cuda_to_cpu_cl	pointer to a codelet which converts from CUDA to CPU

57.5.2.13 struct starpu_multiformat_interface

todo

Data Fields

enum starpu_data_interface_id	id	
void *	cpu_ptr	
void *	cuda_ptr	
void *	hip_ptr	
void *	opencl_ptr	
uint32_t	nx	
struct starpu_multiformat_data_interface_ops *	ops	

57.5.3 Macro Definition Documentation

57.5.3.1 STARPU_MATRIX_GET_PTR

```
#define STARPU_MATRIX_GET_PTR(
                interface )
```

Return a pointer to the matrix designated by `interface`, valid on CPUs and CUDA devices only. For OpenCL devices, the device handle and offset need to be used instead.

57.5.3.2 STARPU_MATRIX_GET_DEV_HANDLE

```
#define STARPU_MATRIX_GET_DEV_HANDLE(
                interface )
```

Return a device handle for the matrix designated by `interface`, to be used with OpenCL. The offset returned by STARPU_MATRIX_GET_OFFSET has to be used in addition to this.

57.5.3.3 STARPU_MATRIX_GET_OFFSET

```
#define STARPU_MATRIX_GET_OFFSET(
            interface )
```

Return the offset in the matrix designated by `interface`, to be used with the device handle.

57.5.3.4 STARPU_MATRIX_GET_NX

```
#define STARPU_MATRIX_GET_NX(
            interface )
```

Return the number of elements on the x-axis of the matrix designated by `interface`.

57.5.3.5 STARPU_MATRIX_GET_NY

```
#define STARPU_MATRIX_GET_NY(
            interface )
```

Return the number of elements on the y-axis of the matrix designated by `interface`.

57.5.3.6 STARPU_MATRIX_GET_LD

```
#define STARPU_MATRIX_GET_LD(
            interface )
```

Return the number of elements between each row of the matrix designated by `interface`. May be equal to nx when there is no padding.

57.5.3.7 STARPU_MATRIX_GET_ELEMSIZE

```
#define STARPU_MATRIX_GET_ELEMSIZE(
            interface )
```

Return the size of the elements registered into the matrix designated by `interface`.

57.5.3.8 STARPU_MATRIX_GET_ALLOCSIZE

```
#define STARPU_MATRIX_GET_ALLOCSIZE(
            interface )
```

Return the allocated size of the matrix designated by `interface`.

57.5.3.9 STARPU_MATRIX_SET_NX

```
#define STARPU_MATRIX_SET_NX(
            interface,
            newnx )
```

Set the number of elements on the x-axis of the matrix designated by `interface`.

57.5.3.10 STARPU_MATRIX_SET_NY

```
#define STARPU_MATRIX_SET_NY(
            interface,
            newny )
```

Set the number of elements on the y-axis of the matrix designated by `interface`.

57.5.3.11 STARPU_MATRIX_SET_LD

```
#define STARPU_MATRIX_SET_LD(
            interface,
            newld )
```

Set the number of elements between each row of the matrix designated by `interface`. May be set to the same value as nx when there is no padding.

57.5.3.12 STARPU_COO_GET_COLUMNS

```
#define STARPU_COO_GET_COLUMNS(
            interface )
```

Return a pointer to the column array of the matrix designated by `interface`.

57.5.3.13 STARPU_COO_GET_COLUMNS_DEV_HANDLE

```
#define STARPU_COO_GET_COLUMNS_DEV_HANDLE(
            interface )
```

Return a device handle for the column array of the matrix designated by `interface`, to be used with OpenCL. The offset returned by STARPU_COO_GET_OFFSET has to be used in addition to this.

57.5.3.14 STARPU_COO_GET_ROWS

```
#define STARPU_COO_GET_ROWS(
            interface )
```

Return a pointer to the rows array of the matrix designated by `interface`.

57.5.3.15 STARPU_COO_GET_ROWS_DEV_HANDLE

```
#define STARPU_COO_GET_ROWS_DEV_HANDLE(
            interface )
```

Return a device handle for the row array of the matrix designated by `interface`, to be used on OpenCL. The offset returned by STARPU_COO_GET_OFFSET has to be used in addition to this.

57.5.3.16 STARPU_COO_GET_VALUES

```
#define STARPU_COO_GET_VALUES(
            interface )
```

Return a pointer to the values array of the matrix designated by `interface`.

57.5.3.17 STARPU_COO_GET_VALUES_DEV_HANDLE

```
#define STARPU_COO_GET_VALUES_DEV_HANDLE(
            interface )
```

Return a device handle for the value array of the matrix designated by `interface`, to be used on OpenCL. The offset returned by STARPU_COO_GET_OFFSET has to be used in addition to this.

57.5.3.18 STARPU_COO_GET_OFFSET

```
#define STARPU_COO_GET_OFFSET
```

Return the offset in the arrays of the COO matrix designated by `interface`.

57.5.3.19 STARPU_COO_GET_NX

```
#define STARPU_COO_GET_NX(
            interface )
```

Return the number of elements on the x-axis of the matrix designated by `interface`.

57.5.3.20 STARPU_COO_GET_NY

```
#define STARPU_COO_GET_NY(
            interface )
```

Return the number of elements on the y-axis of the matrix designated by `interface`.

57.5.3.21 STARPU_COO_GET_NVALUES

```
#define STARPU_COO_GET_NVALUES(
            interface )
```

Return the number of values registered in the matrix designated by `interface`.

57.5.3.22 STARPU_COO_GET_ELEMSIZE

```
#define STARPU_COO_GET_ELEMSIZE(
            interface )
```

Return the size of the elements registered into the matrix designated by `interface`.

57.5.3.23 STARPU_BLOCK_GET_PTR

```
#define STARPU_BLOCK_GET_PTR(
            interface )
```

Return a pointer to the block designated by `interface`.

57.5.3.24 STARPU_BLOCK_GET_DEV_HANDLE

```
#define STARPU_BLOCK_GET_DEV_HANDLE(
            interface )
```

Return a device handle for the block designated by `interface`, to be used on OpenCL. The offset returned by STARPU_BLOCK_GET_OFFSET has to be used in addition to this.

57.5.3.25 STARPU_BLOCK_GET_OFFSET

```
#define STARPU_BLOCK_GET_OFFSET(
            interface )
```

Return the offset in the block designated by `interface`, to be used with the device handle.

57.5.3.26 STARPU_BLOCK_GET_NX

```
#define STARPU_BLOCK_GET_NX(
            interface )
```

Return the number of elements on the x-axis of the block designated by `interface`.

57.5.3.27 STARPU_BLOCK_GET_NY

```
#define STARPU_BLOCK_GET_NY(
            interface )
```

Return the number of elements on the y-axis of the block designated by `interface`.

57.5.3.28 STARPU_BLOCK_GET_NZ

```
#define STARPU_BLOCK_GET_NZ(
            interface )
```

Return the number of elements on the z-axis of the block designated by `interface`.

57.5.3.29 STARPU_BLOCK_GET_LDY

```
#define STARPU_BLOCK_GET_LDY(
            interface )
```

Return the number of elements between each row of the block designated by `interface`. May be equal to nx when there is no padding.

57.5.3.30 STARPU_BLOCK_GET_LDZ

```
#define STARPU_BLOCK_GET_LDZ(
            interface )
```

Return the number of elements between each z plane of the block designated by `interface`. May be equal to nx∗ny when there is no padding.

57.5.3.31 STARPU_BLOCK_GET_ELEMSIZE

```
#define STARPU_BLOCK_GET_ELEMSIZE(
            interface )
```

Return the size of the elements of the block designated by `interface`.

57.5.3.32 STARPU_TENSOR_GET_PTR

```
#define STARPU_TENSOR_GET_PTR(
            interface )
```

Return a pointer to the tensor designated by `interface`.

57.5.3.33 STARPU_TENSOR_GET_DEV_HANDLE

```
#define STARPU_TENSOR_GET_DEV_HANDLE(
            interface )
```

Return a device handle for the tensor designated by `interface`, to be used on OpenCL. The offset returned by STARPU_TENSOR_GET_OFFSET has to be used in addition to this.

57.5.3.34 STARPU_TENSOR_GET_OFFSET

```
#define STARPU_TENSOR_GET_OFFSET(
            interface )
```

Return the offset in the tensor designated by `interface`, to be used with the device handle.

57.5.3.35 STARPU_TENSOR_GET_NX

```
#define STARPU_TENSOR_GET_NX(
            interface )
```

Return the number of elements on the x-axis of the tensor designated by `interface`.

57.5.3.36 STARPU_TENSOR_GET_NY

```
#define STARPU_TENSOR_GET_NY(
            interface )
```

Return the number of elements on the y-axis of the tensor designated by `interface`.

57.5.3.37 STARPU_TENSOR_GET_NZ

```
#define STARPU_TENSOR_GET_NZ(
            interface )
```

Return the number of elements on the z-axis of the tensor designated by `interface`.

57.5.3.38 STARPU_TENSOR_GET_NT

```
#define STARPU_TENSOR_GET_NT(
            interface )
```

Return the number of elements on the t-axis of the tensor designated by `interface`.

57.5.3.39 STARPU_TENSOR_GET_LDY

```
#define STARPU_TENSOR_GET_LDY(
            interface )
```

Return the number of elements between each row of the tensor designated by `interface`. May be equal to nx when there is no padding.

57.5.3.40 STARPU_TENSOR_GET_LDZ

```
#define STARPU_TENSOR_GET_LDZ(
            interface )
```

Return the number of elements between each z plane of the tensor designated by `interface`. May be equal to nx∗ny when there is no padding.

57.5.3.41 STARPU_TENSOR_GET_LDT

```
#define STARPU_TENSOR_GET_LDT(
            interface )
```

Return the number of elements between each t cubes of the tensor designated by `interface`. May be equal to nx∗ny∗nz when there is no padding.

57.5.3.42 STARPU_TENSOR_GET_ELEMSIZE

```
#define STARPU_TENSOR_GET_ELEMSIZE(
            interface )
```

Return the size of the elements of the tensor designated by `interface`.

57.5.3.43 STARPU_NDIM_GET_PTR

```
#define STARPU_NDIM_GET_PTR(
            interface )
```

Return a pointer to the ndim array designated by `interface`.

57.5.3.44 STARPU_NDIM_GET_DEV_HANDLE

```
#define STARPU_NDIM_GET_DEV_HANDLE(
            interface )
```

Return a device handle for the ndim array designated by `interface`, to be used on OpenCL. The offset returned by STARPU_NDIM_GET_OFFSET has to be used in addition to this.

57.5.3.45 STARPU_NDIM_GET_OFFSET

```
#define STARPU_NDIM_GET_OFFSET(
            interface )
```

Return the offset in the ndim designated by `interface`, to be used with the device handle.

57.5.3.46 STARPU_NDIM_GET_NN

```
#define STARPU_NDIM_GET_NN(
            interface )
```

Return the number of elements on each dimension of the ndim array designated by `interface`.

57.5.3.47 STARPU_NDIM_GET_LDN

```
#define STARPU_NDIM_GET_LDN(
            interface )
```

Return the number of elements between each two units on each dimension of the ndim array designated by `interface`. May be equal to nx when there is no padding.

57.5.3.48 STARPU_NDIM_GET_NDIM

```
#define STARPU_NDIM_GET_NDIM(
            interface )
```

Return the dimension size of the ndim array designated by `interface`.

57.5.3.49 STARPU_NDIM_GET_ELEMSIZE

```
#define STARPU_NDIM_GET_ELEMSIZE(
            interface )
```

Return the size of the elements of the ndim array designated by `interface`.

57.5.3.50 STARPU_VECTOR_GET_PTR

```
#define STARPU_VECTOR_GET_PTR(
            interface )
```

Return a pointer to the array designated by `interface`, valid on CPUs and CUDA only. For OpenCL, the device handle and offset need to be used instead.

57.5.3.51 STARPU_VECTOR_GET_DEV_HANDLE

```
#define STARPU_VECTOR_GET_DEV_HANDLE(
            interface )
```

Return a device handle for the array designated by `interface`, to be used with OpenCL. the offset returned by STARPU_VECTOR_GET_OFFSET has to be used in addition to this.

57.5.3.52 STARPU_VECTOR_GET_OFFSET

```
#define STARPU_VECTOR_GET_OFFSET(
            interface )
```

Return the offset in the array designated by `interface`, to be used with the device handle.

57.5.3.53 STARPU_VECTOR_GET_NX

```
#define STARPU_VECTOR_GET_NX(
            interface )
```

Return the number of elements registered into the array designated by `interface`.

57.5.3.54 STARPU_VECTOR_GET_ELEMSIZE

```
#define STARPU_VECTOR_GET_ELEMSIZE(
            interface )
```

Return the size of each element of the array designated by `interface`.

57.5.3.55 STARPU_VECTOR_GET_ALLOCSIZE

```
#define STARPU_VECTOR_GET_ALLOCSIZE(
            interface )
```

Return the size of each element of the array designated by `interface`.

57.5.3.56 STARPU_VECTOR_GET_SLICE_BASE

```
#define STARPU_VECTOR_GET_SLICE_BASE(
            interface )
```

Return the OpenMP slice base annotation of each element of the array designated by `interface`.

57.5.3.57 STARPU_VECTOR_SET_NX

```
#define STARPU_VECTOR_SET_NX(
            interface,
            newnx )
```

Set the number of elements registered into the array designated by `interface`.

57.5.3.58 STARPU_VARIABLE_GET_PTR

```
#define STARPU_VARIABLE_GET_PTR(
            interface )
```

Return a pointer to the variable designated by `interface`.

57.5.3.59 STARPU_VARIABLE_GET_OFFSET

```
#define STARPU_VARIABLE_GET_OFFSET(
            interface )
```

Return the offset in the variable designated by `interface`, to be used with the device handle.

57.5.3.60 STARPU_VARIABLE_GET_ELEMSIZE

```
#define STARPU_VARIABLE_GET_ELEMSIZE(
            interface )
```

Return the size of the variable designated by `interface`.

57.5.3.61 STARPU_VARIABLE_GET_DEV_HANDLE

```
#define STARPU_VARIABLE_GET_DEV_HANDLE(
            interface )
```

Return a device handle for the variable designated by `interface`, to be used with OpenCL. The offset returned by STARPU_VARIABLE_GET_OFFSET has to be used in addition to this.

57.5.3.62 STARPU_CSR_GET_NNZ

```
#define STARPU_CSR_GET_NNZ(
            interface )
```

Return the number of non-zero values in the matrix designated by `interface`.

57.5.3.63 STARPU_CSR_GET_NROW

```
#define STARPU_CSR_GET_NROW(
            interface )
```

Return the size of the row pointer array of the matrix designated by `interface`.

57.5.3.64 STARPU_CSR_GET_NZVAL

```
#define STARPU_CSR_GET_NZVAL(
            interface )
```

Return a pointer to the non-zero values of the matrix designated by `interface`.

57.5.3.65 STARPU_CSR_GET_NZVAL_DEV_HANDLE

```
#define STARPU_CSR_GET_NZVAL_DEV_HANDLE(
            interface )
```

Return a device handle for the array of non-zero values in the matrix designated by `interface`. The offset returned by STARPU_CSR_GET_OFFSET has to used in addition to this.

57.5.3.66 STARPU_CSR_GET_COLIND

```
#define STARPU_CSR_GET_COLIND(
            interface )
```

Return a pointer to the column index of the matrix designated by `interface`.

57.5.3.67 STARPU_CSR_GET_RAM_COLIND

```
#define STARPU_CSR_GET_RAM_COLIND(
            interface )
```

Return a RAM pointer to the column index of the matrix designated by `interface`.

57.5.3.68 STARPU_CSR_GET_COLIND_DEV_HANDLE

```
#define STARPU_CSR_GET_COLIND_DEV_HANDLE(
            interface )
```

Return a device handle for the column index of the matrix designated by `interface`. The offset returned by STARPU_CSR_GET_OFFSET has to be used in addition to this.

57.5.3.69 STARPU_CSR_GET_ROWPTR

```
#define STARPU_CSR_GET_ROWPTR(
            interface )
```

Return a pointer to the row pointer array of the matrix designated by `interface`.

57.5.3.70 STARPU_CSR_GET_RAM_ROWPTR

```
#define STARPU_CSR_GET_RAM_ROWPTR(
            interface )
```

Return a RAM pointer to the row pointer array of the matrix designated by `interface`.

57.5.3.71 STARPU_CSR_GET_ROWPTR_DEV_HANDLE

```
#define STARPU_CSR_GET_ROWPTR_DEV_HANDLE(
            interface )
```

Return a device handle for the row pointer array of the matrix designated by `interface`. The offset returned by STARPU_CSR_GET_OFFSET has to be used in addition to this.

57.5.3.72 STARPU_CSR_GET_OFFSET

```
#define STARPU_CSR_GET_OFFSET
```

Return the offset in the arrays (colind, rowptr, nzval) of the matrix designated by `interface`, to be used with the device handles.

57.5.3.73 STARPU_CSR_GET_FIRSTENTRY

```
#define STARPU_CSR_GET_FIRSTENTRY(
            interface )
```

Return the index at which all arrays (the column indexes, the row pointers...) of the `interface` start.

57.5.3.74 STARPU_CSR_GET_ELEMSIZE

```
#define STARPU_CSR_GET_ELEMSIZE(
            interface )
```

Return the size of the elements registered into the matrix designated by `interface`.

57.5.3.75 STARPU_BCSR_GET_NNZ

```
#define STARPU_BCSR_GET_NNZ(
            interface )
```

Return the number of non-zero values in the matrix designated by `interface`.

57.5.3.76 STARPU_BCSR_GET_NROW

```
#define STARPU_BCSR_GET_NROW(
            interface )
```

Return the number of block rows in the matrix designated by `interface`.

57.5.3.77 STARPU_BCSR_GET_NZVAL

```
#define STARPU_BCSR_GET_NZVAL(
            interface )
```

Return a pointer to the non-zero values of the matrix designated by `interface`.

57.5.3.78 STARPU_BCSR_GET_NZVAL_DEV_HANDLE

```
#define STARPU_BCSR_GET_NZVAL_DEV_HANDLE(
            interface )
```

Return a device handle for the array of non-zero values in the matrix designated by `interface`. The offset returned by STARPU_BCSR_GET_OFFSET has to be used in addition to this.

57.5.3.79 STARPU_BCSR_GET_COLIND

```
#define STARPU_BCSR_GET_COLIND(
            interface )
```

Return a pointer to the column index of the matrix designated by `interface`.

57.5.3.80 STARPU_BCSR_GET_RAM_COLIND

```
#define STARPU_BCSR_GET_RAM_COLIND(
            interface )
```

Return a RAM pointer to the column index of the matrix designated by `interface`.

57.5.3.81 STARPU_BCSR_GET_COLIND_DEV_HANDLE

```
#define STARPU_BCSR_GET_COLIND_DEV_HANDLE(
            interface )
```

Return a device handle for the column index of the matrix designated by `interface`. The offset returned by STARPU_BCSR_GET_OFFSET has to be used in addition to this.

57.5.3.82 STARPU_BCSR_GET_ROWPTR

```
#define STARPU_BCSR_GET_ROWPTR(
            interface )
```

Return a pointer to the row pointer array of the matrix designated by `interface`.

57.5.3.83 STARPU_BCSR_GET_RAM_ROWPTR

```
#define STARPU_BCSR_GET_RAM_ROWPTR(
            interface )
```

Return a RAM pointer to the row pointer array of the matrix designated by `interface`.

57.5.3.84 STARPU_BCSR_GET_ROWPTR_DEV_HANDLE

```
#define STARPU_BCSR_GET_ROWPTR_DEV_HANDLE(
            interface )
```

Return a device handle for the row pointer array of the matrix designated by `interface`. The offset returned by STARPU_BCSR_GET_OFFSET has to be used in addition to this.

57.5.3.85 STARPU_BCSR_GET_FIRSTENTRY

```
#define STARPU_BCSR_GET_FIRSTENTRY(
            interface )
```

Return the base of the indexing (0 or 1 usually) in the matrix designated by `interface`.

57.5.3.86 STARPU_BCSR_GET_R

```
#define STARPU_BCSR_GET_R(
            interface )
```

Return the height of blocks in the matrix designated by `interface`.

57.5.3.87 STARPU_BCSR_GET_C

```
#define STARPU_BCSR_GET_C(
            interface )
```

Return the width of blocks in the matrix designated by `interface`.

57.5.3.88 STARPU_BCSR_GET_ELEMSIZE

```
#define STARPU_BCSR_GET_ELEMSIZE(
            interface )
```

Return the size of elements in the matrix designated by `interface`.

57.5.3.89 STARPU_BCSR_GET_OFFSET

```
#define STARPU_BCSR_GET_OFFSET
```

Return the offset in the arrays (coling, rowptr, nzval) of the matrix designated by `interface`, to be used with the device handles.

57.5.3.90 STARPU_MULTIFORMAT_GET_CPU_PTR

```
#define STARPU_MULTIFORMAT_GET_CPU_PTR(
            interface )
```

Return the local pointer to the data with CPU format.

57.5.3.91 STARPU_MULTIFORMAT_GET_CUDA_PTR

```
#define STARPU_MULTIFORMAT_GET_CUDA_PTR(
            interface )
```

Return the local pointer to the data with CUDA format.

57.5.3.92 STARPU_MULTIFORMAT_GET_HIP_PTR

```
#define STARPU_MULTIFORMAT_GET_HIP_PTR(
            interface )
```

Return the local pointer to the data with HIP format.

57.5.3.93 STARPU_MULTIFORMAT_GET_OPENCL_PTR

```
#define STARPU_MULTIFORMAT_GET_OPENCL_PTR(
            interface )
```

Return the local pointer to the data with OpenCL format.

57.5.3.94 STARPU_MULTIFORMAT_GET_NX

```
#define STARPU_MULTIFORMAT_GET_NX(
            interface )
```

Return the number of elements in the data.

57.5.4 Enumeration Type Documentation

57.5.4.1 starpu_data_interface_id

`enum starpu_data_interface_id`

Identifier for all predefined StarPU data interfaces

Enumerator

STARPU_UNKNOWN_INTERFACE_ID	Unknown interface

Enumerator

STARPU_MATRIX_INTERFACE_ID	Identifier for the matrix data interface
STARPU_BLOCK_INTERFACE_ID	Identifier for the block data interface
STARPU_VECTOR_INTERFACE_ID	Identifier for the vector data interface
STARPU_CSR_INTERFACE_ID	Identifier for the CSR data interface
STARPU_BCSR_INTERFACE_ID	Identifier for the BCSR data interface
STARPU_VARIABLE_INTERFACE_ID	Identifier for the variable data interface
STARPU_VOID_INTERFACE_ID	Identifier for the void data interface
STARPU_MULTIFORMAT_INTERFACE_ID	Identifier for the multiformat data interface
STARPU_COO_INTERFACE_ID	Identifier for the COO data interface
STARPU_TENSOR_INTERFACE_ID	Identifier for the tensor data interface
STARPU_NDIM_INTERFACE_ID	Identifier for the ndim array data interface
STARPU_MAX_INTERFACE_ID	Maximum number of data interfaces

57.5.5 Function Documentation

57.5.5.1 starpu_data_register()

```
void starpu_data_register (
            starpu_data_handle_t * handleptr,
            int home_node,
            void * data_interface,
            struct starpu_data_interface_ops * ops )
```

Register a piece of data into the handle located at the `handleptr` address. The `data_interface` buffer contains the initial description of the data in the `home_node`. The `ops` argument is a pointer to a structure describing the different methods used to manipulate this type of interface. See starpu_data_interface_ops for more details on this structure. If `home_node` is -1, StarPU will automatically allocate the memory when it is used for the first time in write-only mode. Once such data handle has been automatically allocated, it is possible to access it using any access mode. Note that StarPU supplies a set of predefined types of interface (e.g. vector or matrix) which can be registered by the means of helper functions (e.g. starpu_vector_data_register() or starpu_matrix_data_register()). See Data registration for more details.

57.5.5.2 starpu_data_register_ops()

```
void starpu_data_register_ops (
            struct starpu_data_interface_ops * ops )
```

Register the given data interface operations. If the field starpu_data_interface_ops::field is set to STARPU_UNKNOWN_INTERFACE_ then a new identifier will be set by calling starpu_data_interface_get_next_id(). The function is automatically called when registering a piece of data with starpu_data_register(). It is only necessary to call it beforehand for some specific cases (such as the usmaster slave mode).

57.5.5.3 starpu_data_ptr_register()

```
void starpu_data_ptr_register (
            starpu_data_handle_t handle,
            unsigned node )
```

Register that a buffer for `handle` on `node` will be set. This is typically used by starpu_*_ptr_register helpers before setting the interface pointers for this node, to tell the core that that is now allocated. See Pointers inside the data interface for more details.

57.5.5.4 starpu_data_register_same()

```
void starpu_data_register_same (
            starpu_data_handle_t * handledst,
            starpu_data_handle_t handlesrc )
```

Register a new piece of data into the handle `handledst` with the same interface as the handle `handlesrc`. See Data handles helpers for more details.

57.5.5.5 starpu_data_handle_to_pointer()

```
void * starpu_data_handle_to_pointer (
            starpu_data_handle_t handle,
            unsigned node )
```

Return the pointer associated with `handle` on node `node` or `NULL` if handle's interface does not support this operation or data for this `handle` is not allocated on that `node`. See Handles data buffer pointers for more details.

57.5.5.6 starpu_data_get_local_ptr()

```
void * starpu_data_get_local_ptr (
            starpu_data_handle_t handle )
```

Return the local pointer associated with `handle` or `NULL` if `handle's` interface does not have any data allocated locally. See Handles data buffer pointers for more details.

57.5.5.7 starpu_data_get_interface_on_node()

```
void * starpu_data_get_interface_on_node (
            starpu_data_handle_t handle,
            unsigned memory_node )
```

Return the interface associated with `handle` on `memory_node`. See Data pack/peek/unpack for more details.

57.5.5.8 starpu_data_get_interface_id()

```
enum starpu_data_interface_id starpu_data_get_interface_id (
            starpu_data_handle_t handle )
```

Return the unique identifier of the interface associated with the given `handle`. See Helpers for more details.

57.5.5.9 starpu_data_pack_node()

```
int starpu_data_pack_node (
            starpu_data_handle_t handle,
            unsigned node,
            void ** ptr,
            starpu_ssize_t * count )
```

Execute the packing operation of the interface of the data registered at `handle` (see starpu_data_interface_ops). This packing operation must allocate a buffer large enough at `ptr` on node `node` and copy into the newly allocated buffer the data associated to `handle`. `count` will be set to the size of the allocated buffer. If `ptr` is `NULL`, the function should not copy the data in the buffer but just set `count` to the size of the buffer which would have been allocated. The special value -1 indicates the size is yet unknown. See Data handles helpers for more details.

57.5.5.10 starpu_data_pack()

```
int starpu_data_pack (
            starpu_data_handle_t handle,
            void ** ptr,
            starpu_ssize_t * count )
```

Like starpu_data_pack_node(), but for the local memory node. See Data handles helpers for more details.

57.5.5.11 starpu_data_peek_node()

```
int starpu_data_peek_node (
            starpu_data_handle_t handle,
            unsigned node,
            void * ptr,
            size_t count )
```

Read in handle's `node` replicate the data located at `ptr` of size `count` as described by the interface of the data. The interface registered at `handle` must define a peeking operation (see starpu_data_interface_ops). See Data handles helpers for more details.

57.5.5.12 starpu_data_peek()

```
int starpu_data_peek (
            starpu_data_handle_t handle,
            void * ptr,
            size_t count )
```

Read in handle's local replicate the data located at `ptr` of size `count` as described by the interface of the data. The interface registered at `handle` must define a peeking operation (see starpu_data_interface_ops). See Data handles helpers for more details.

57.5.5.13 starpu_data_unpack_node()

```
int starpu_data_unpack_node (
            starpu_data_handle_t handle,
            unsigned node,
            void * ptr,
            size_t count )
```

Unpack in handle the data located at `ptr` of size `count` allocated on node `node` as described by the interface of the data. The interface registered at `handle` must define an unpacking operation (see starpu_data_interface_ops). See Data handles helpers for more details.

57.5.5.14 starpu_data_unpack()

```
int starpu_data_unpack (
            starpu_data_handle_t handle,
            void * ptr,
            size_t count )
```

Unpack in handle the data located at `ptr` of size `count` as described by the interface of the data. The interface registered at `handle` must define a unpacking operation (see starpu_data_interface_ops). See Data handles helpers for more details.

57.5.5.15 starpu_data_get_size()

```
size_t starpu_data_get_size (
            starpu_data_handle_t handle )
```

Return the size of the data associated with `handle`. See Data handles helpers for more details.

57.5.5.16 starpu_data_get_alloc_size()

```
size_t starpu_data_get_alloc_size (
            starpu_data_handle_t handle )
```

Return the size of the allocated data associated with `handle`. See Data handles helpers for more details.

57.5.5.17 starpu_data_get_max_size()

```
starpu_ssize_t starpu_data_get_max_size (
            starpu_data_handle_t handle )
```

Return the maximum size that the `handle` data may need to increase to. See Data handles helpers for more details.

57.5.5.18 starpu_data_get_home_node()

```
int starpu_data_get_home_node (
            starpu_data_handle_t handle )
```

See Data handles helpers for more details.

57.5.5.19 starpu_data_print()

```
void starpu_data_print (
            starpu_data_handle_t handle,
            unsigned node,
            FILE * stream )
```

Print basic information on `handle` on `node`. See Data handles helpers for more details.

57.5.5.20 starpu_data_interface_get_next_id()

```
int starpu_data_interface_get_next_id (
            void )
```

Return the next available id for a newly created data interface (Defining A New Data Interface).

57.5.5.21 starpu_interface_copy()

```
int starpu_interface_copy (
            uintptr_t src,
            size_t src_offset,
            unsigned src_node,
            uintptr_t dst,
            size_t dst_offset,
            unsigned dst_node,
            size_t size,
            void * async_data )
```

Copy `size` bytes from byte offset `src_offset` of `src` on `src_node` to byte offset `dst_offset` of `dst` on `dst_node`. This is to be used in the starpu_data_copy_methods::any_to_any copy method, which is provided with `async_data` to be passed to starpu_interface_copy(). this returns `-EAGAIN` if the transfer is still ongoing, or 0 if the transfer is already completed.
See Data copy for more details.

57.5.5.22 starpu_interface_copy2d()

```
int starpu_interface_copy2d (
            uintptr_t src,
            size_t src_offset,
            unsigned src_node,
            uintptr_t dst,
            size_t dst_offset,
            unsigned dst_node,
            size_t blocksize,
            size_t numblocks,
            size_t ld_src,
            size_t ld_dst,
            void * async_data )
```

Copy `numblocks` blocks of `blocksize` bytes from byte offset `src_offset` of `src` on `src_node` to byte offset `dst_offset` of `dst` on `dst_node`.
The blocks start at addresses which are ld_src (resp. ld_dst) bytes apart in the source (resp. destination) interface.
If blocksize == ld_src == ld_dst, the transfer is optimized into a single starpu_interface_copy call.
This is to be used in the starpu_data_copy_methods::any_to_any copy method for 2D data, which is provided with `async_data` to be passed to starpu_interface_copy(). this returns `-EAGAIN` if the transfer is still ongoing, or 0 if the transfer is already completed.
See Data copy for more details.

57.5.5.23 starpu_interface_copy3d()

```
int starpu_interface_copy3d (
            uintptr_t src,
            size_t src_offset,
            unsigned src_node,
            uintptr_t dst,
            size_t dst_offset,
            unsigned dst_node,
            size_t blocksize,
            size_t numblocks1,
            size_t ld1_src,
            size_t ld1_dst,
            size_t numblocks2,
            size_t ld2_src,
            size_t ld2_dst,
            void * async_data )
```

Copy `numblocks_1 * numblocks_2` blocks of `blocksize` bytes from byte offset `src_offset` of `src` on `src_node` to byte offset `dst_offset` of `dst` on `dst_node`.
The blocks are grouped by `numblocks_1` blocks whose start addresses are ld1_src (resp. ld1_dst) bytes apart in the source (resp. destination) interface.
Such groups are grouped by numblocks_2 groups whose start addresses are ld2_src (resp. ld2_dst) bytes apart in the source (resp. destination) interface.
If the blocks are contiguous, the transfers will be optimized.
This is to be used in the starpu_data_copy_methods::any_to_any copy method for 3D data, which is provided with `async_data` to be passed to starpu_interface_copy(). this returns `-EAGAIN` if the transfer is still ongoing, or 0 if the transfer is already completed.
See Data copy for more details.

57.5.5.24 starpu_interface_copy4d()

```
int starpu_interface_copy4d (
            uintptr_t src,
            size_t src_offset,
            unsigned src_node,
            uintptr_t dst,
            size_t dst_offset,
            unsigned dst_node,
            size_t blocksize,
            size_t numblocks1,
            size_t ld1_src,
            size_t ld1_dst,
            size_t numblocks2,
            size_t ld2_src,
            size_t ld2_dst,
            size_t numblocks3,
            size_t ld3_src,
            size_t ld3_dst,
            void * async_data )
```

Copy `numblocks_1 * numblocks_2 * numblocks_3` blocks of `blocksize` bytes from byte offset `src_offset` of `src` on `src_node` to byte offset `dst_offset` of `dst` on `dst_node`.
The blocks are grouped by `numblocks_1` blocks whose start addresses are ld1_src (resp. ld1_dst) bytes apart in the source (resp. destination) interface.
Such groups are grouped by numblocks_2 groups whose start addresses are ld2_src (resp. ld2_dst) bytes apart in the source (resp. destination) interface.
Such groups are grouped by numblocks_3 groups whose start addresses are ld3_src (resp. ld3_dst) bytes apart in the source (resp. destination) interface.
If the blocks are contiguous, the transfers will be optimized.
This is to be used in the starpu_data_copy_methods::any_to_any copy method for 4D data, which is provided with

`async_data` to be passed to starpu_interface_copy(). this returns `−EAGAIN` if the transfer is still ongoing, or 0 if the transfer is already completed.
See Data copy for more details.

57.5.5.25 starpu_interface_copynd()

```
int starpu_interface_copynd (
            uintptr_t src,
            size_t src_offset,
            unsigned src_node,
            uintptr_t dst,
            size_t dst_offset,
            unsigned dst_node,
            size_t elemsize,
            size_t ndim,
            uint32_t * nn,
            uint32_t * ldn_src,
            uint32_t * ldn_dst,
            void * async_data )
```

Copy `nn`[1] ∗ `nn`[2]...∗ `nn`[ndim-1] blocks of `nn`[0] ∗ `elemsize` bytes from byte offset `src_offset` of `src` on `src_node` to byte offset `dst_offset` of `dst` on `dst_node`.
The blocks are grouped by `nn`[i] blocks (i = 1, 2, ... ndim-1) whose start addresses are ldn_src[i] ∗ `elemsize` (resp. ld1_dst[i] ∗ `elemsize`) bytes apart in the source (resp. destination) interface.
If the blocks are contiguous, the transfers will be optimized.
This is to be used in the starpu_data_copy_methods::any_to_any copy method for Ndim data, which is provided with `async_data` to be passed to starpu_interface_copy(). this returns `−EAGAIN` if the transfer is still ongoing, or 0 if the transfer is already completed.
See Data copy for more details.

57.5.5.26 starpu_interface_start_driver_copy_async()

```
void starpu_interface_start_driver_copy_async (
            unsigned src_node,
            unsigned dst_node,
            double * start )
```

When an asynchronous implementation of the data transfer is implemented, the call to the underlying CUDA, OpenCL, etc. call should be surrounded by calls to starpu_interface_start_driver_copy_async() and starpu_interface_end_driver_copy_async(), so that it is recorded in offline execution traces, and the timing of the submission is checked. `start` must point to a variable whose value will be passed unchanged to starpu_interface_end_driver_copy_async().
See Data copy for more details.

57.5.5.27 starpu_interface_end_driver_copy_async()

```
void starpu_interface_end_driver_copy_async (
            unsigned src_node,
            unsigned dst_node,
            double start )
```

See starpu_interface_start_driver_copy_async(). See Data copy for more details.

57.5.5.28 starpu_interface_data_copy()

```
void starpu_interface_data_copy (
            unsigned src_node,
            unsigned dst_node,
            size_t size )
```

Record in offline execution traces the copy of `size` bytes from node `src_node` to node `dst_node`. See Data copy for more details.

57.5.5.29 starpu_malloc_on_node_flags()

```
uintptr_t starpu_malloc_on_node_flags (
            unsigned dst_node,
            size_t size,
            int flags )
```

Allocate `size` bytes on node `dst_node` with the given allocation `flags` (such as STARPU_MALLOC_PINNED, STARPU_MALLOC_COUNT, etc.). This returns 0 if allocation failed, the allocation method should then return `-ENOMEM` as allocated size. Deallocation must be done with starpu_free_on_node_flags().
See Data Interface with Variable Size for more details.

57.5.5.30 starpu_malloc_on_node()

```
uintptr_t starpu_malloc_on_node (
            unsigned dst_node,
            size_t size )
```

Allocate `size` bytes on node `dst_node` with the default allocation flags. This returns 0 if allocation failed, the allocation method should then return `-ENOMEM` as allocated size. Deallocation must be done with starpu_free_on_node().
See Data allocation for more details.

57.5.5.31 starpu_free_on_node_flags()

```
void starpu_free_on_node_flags (
            unsigned dst_node,
            uintptr_t addr,
            size_t size,
            int flags )
```

Free `addr` of `size` bytes on node `dst_node` which was previously allocated with starpu_malloc_on_node_flags() with the given allocation `flags`.
See Data Interface with Variable Size for more details.

57.5.5.32 starpu_free_on_node()

```
void starpu_free_on_node (
            unsigned dst_node,
            uintptr_t addr,
            size_t size )
```

Free `addr` of `size` bytes on node `dst_node` which was previously allocated with starpu_malloc_on_node().
See Data allocation for more details.

57.5.5.33 starpu_malloc_on_node_set_default_flags()

```
void starpu_malloc_on_node_set_default_flags (
            unsigned node,
            int flags )
```

Define the default flags for allocations performed by starpu_malloc_on_node() and starpu_free_on_node(). The default is STARPU_MALLOC_PINNED | STARPU_MALLOC_COUNT. See How to Limit Memory Used By StarPU And Cache Buffer All
for more details.

57.5.5.34 starpu_interface_map()

```
uintptr_t starpu_interface_map (
            uintptr_t src,
            size_t src_offset,
            unsigned src_node,
            unsigned dst_node,
            size_t size,
            int * ret )
```

Used to set starpu_data_interface_ops::map_data. See Pointers inside the data interface for more details.

57.5.5.35 starpu_interface_unmap()

```
int starpu_interface_unmap (
            uintptr_t src,
            size_t src_offset,
            unsigned src_node,
            uintptr_t dst,
            unsigned dst_node,
            size_t size )
```

Used to set starpu_data_interface_ops::unmap_data. See Pointers inside the data interface for more details.

57.5.5.36 starpu_interface_update_map()

```
int starpu_interface_update_map (
            uintptr_t src,
            size_t src_offset,
            unsigned src_node,
            uintptr_t dst,
            size_t dst_offset,
            unsigned dst_node,
            size_t size )
```

Used to set starpu_data_interface_ops::update_map. See Pointers inside the data interface for more details.

57.5.5.37 starpu_matrix_data_register()

```
void starpu_matrix_data_register (
            starpu_data_handle_t * handle,
            int home_node,
            uintptr_t ptr,
            uint32_t ld,
            uint32_t nx,
            uint32_t ny,
            size_t elemsize )
```

Register the `nx` x `ny` 2D matrix of `elemsize-byte` elements pointed by `ptr` and initialize `handle` to represent it. `ld` specifies the number of elements between rows. a value greater than `nx` adds padding, which can be useful for alignment purposes.

Here an example of how to use the function.

```
float *matrix;
starpu_data_handle_t matrix_handle;
matrix = (float*)malloc(width * height * sizeof(float));
starpu_matrix_data_register(&matrix_handle, STARPU_MAIN_RAM, (uintptr_t)matrix, width, width, height,
      sizeof(float));
```

See Matrix Data Interface for more details.

57.5.5.38 starpu_matrix_data_register_allocsize()

```
void starpu_matrix_data_register_allocsize (
            starpu_data_handle_t * handle,
            int home_node,
            uintptr_t ptr,
            uint32_t ld,
            uint32_t nx,
            uint32_t ny,
            size_t elemsize,
            size_t allocsize )
```

Similar to starpu_matrix_data_register, but additionally specifies which allocation size should be used instead of the initial nx∗ny∗elemsize.

See Data Interface with Variable Size for more details.

57.5.5.39 starpu_matrix_ptr_register()

```
void starpu_matrix_ptr_register (
            starpu_data_handle_t handle,
            unsigned node,
            uintptr_t ptr,
            uintptr_t dev_handle,
            size_t offset,
            uint32_t ld )
```

Register into the `handle` that to store data on node `node` it should use the buffer located at `ptr`, or device handle `dev_handle` and offset `offset` (for OpenCL, notably), with `ld` elements between rows.

57.5.5.40 starpu_matrix_get_nx()

```
uint32_t starpu_matrix_get_nx (
            starpu_data_handle_t handle )
```

Return the number of elements on the x-axis of the matrix designated by `handle`.

57.5.5.41 starpu_matrix_get_ny()

```
uint32_t starpu_matrix_get_ny (
            starpu_data_handle_t handle )
```

Return the number of elements on the y-axis of the matrix designated by `handle`.

57.5.5.42 starpu_matrix_get_local_ld()

```
uint32_t starpu_matrix_get_local_ld (
            starpu_data_handle_t handle )
```

Return the number of elements between each row of the matrix designated by `handle`. Maybe be equal to nx when there is no padding.

57.5.5.43 starpu_matrix_get_local_ptr()

```
uintptr_t starpu_matrix_get_local_ptr (
            starpu_data_handle_t handle )
```

Return the local pointer associated with `handle`.

57.5.5.44 starpu_matrix_get_elemsize()

```
size_t starpu_matrix_get_elemsize (
            starpu_data_handle_t handle )
```

Return the size of the elements registered into the matrix designated by `handle`.

57.5.5.45 starpu_matrix_get_allocsize()

```
size_t starpu_matrix_get_allocsize (
            starpu_data_handle_t handle )
```

Return the allocated size of the matrix designated by `handle`.

57.5.5.46 starpu_coo_data_register()

```
void starpu_coo_data_register (
            starpu_data_handle_t * handleptr,
            int home_node,
            uint32_t nx,
            uint32_t ny,
            uint32_t n_values,
            uint32_t * columns,
            uint32_t * rows,
            uintptr_t values,
            size_t elemsize )
```

Register the `nx` x `ny` 2D matrix given in the COO format, using the `columns`, `rows`, `values` arrays, which must have `n_values` elements of size `elemsize`. Initialize `handleptr`. See COO Data Interface for more details.

57.5.5.47 starpu_block_data_register()

```
void starpu_block_data_register (
             starpu_data_handle_t * handle,
             int home_node,
             uintptr_t ptr,
             uint32_t ldy,
             uint32_t ldz,
             uint32_t nx,
             uint32_t ny,
             uint32_t nz,
             size_t elemsize )
```

Register the `nx` x `ny` x `nz` 3D matrix of `elemsize` byte elements pointed by `ptr` and initialize `handle` to represent it. Again, `ldy` and `ldz` specify the number of elements between rows and between z planes.

Here an example of how to use the function.

```
float *block;
starpu_data_handle_t block_handle;
block = (float*)malloc(nx*ny*nz*sizeof(float));
starpu_block_data_register(&block_handle, STARPU_MAIN_RAM, (uintptr_t)block, nx, nx*ny, nx, ny, nz,
      sizeof(float));
```

See Block Data Interface for more details.

57.5.5.48 starpu_block_ptr_register()

```
void starpu_block_ptr_register (
             starpu_data_handle_t handle,
             unsigned node,
             uintptr_t ptr,
             uintptr_t dev_handle,
             size_t offset,
             uint32_t ldy,
             uint32_t ldz )
```

Register into the `handle` that to store data on node `node` it should use the buffer located at `ptr`, or device handle `dev_handle` and offset `offset` (for OpenCL, notably), with `ldy` elements between rows and `ldz` elements between z planes.

57.5.5.49 starpu_block_get_nx()

```
uint32_t starpu_block_get_nx (
             starpu_data_handle_t handle )
```

Return the number of elements on the x-axis of the block designated by `handle`.

57.5.5.50 starpu_block_get_ny()

```
uint32_t starpu_block_get_ny (
             starpu_data_handle_t handle )
```

Return the number of elements on the y-axis of the block designated by `handle`.

57.5.5.51 starpu_block_get_nz()

```
uint32_t starpu_block_get_nz (
             starpu_data_handle_t handle )
```

Return the number of elements on the z-axis of the block designated by `handle`.

57.5.5.52 starpu_block_get_local_ldy()

```
uint32_t starpu_block_get_local_ldy (
            starpu_data_handle_t handle )
```

Return the number of elements between each row of the block designated by `handle`, in the format of the current memory node.

57.5.5.53 starpu_block_get_local_ldz()

```
uint32_t starpu_block_get_local_ldz (
            starpu_data_handle_t handle )
```

Return the number of elements between each z plane of the block designated by `handle`, in the format of the current memory node.

57.5.5.54 starpu_block_get_local_ptr()

```
uintptr_t starpu_block_get_local_ptr (
            starpu_data_handle_t handle )
```

Return the local pointer associated with `handle`.

57.5.5.55 starpu_block_get_elemsize()

```
size_t starpu_block_get_elemsize (
            starpu_data_handle_t handle )
```

Return the size of the elements of the block designated by `handle`.

57.5.5.56 starpu_tensor_data_register()

```
void starpu_tensor_data_register (
            starpu_data_handle_t * handle,
            int home_node,
            uintptr_t ptr,
            uint32_t ldy,
            uint32_t ldz,
            uint32_t ldt,
            uint32_t nx,
            uint32_t ny,
            uint32_t nz,
            uint32_t nt,
            size_t elemsize )
```

Register the `nx` x `ny` x `nz` x `nt` 4D tensor of `elemsize` byte elements pointed by `ptr` and initialize `handle` to represent it. Again, `ldy`, `ldz`, and `ldt` specify the number of elements between rows, between z planes and between t cubes.

Here an example of how to use the function.

```
float *tensor;
starpu_data_handle_t tensor_handle;
tensor = (float*)malloc(nx*ny*nz*nt*sizeof(float));
starpu_tensor_data_register(&tensor_handle, STARPU_MAIN_RAM, (uintptr_t)tensor, nx, nx*ny, nx*ny*nz, nx, ny,
      nz, nt, sizeof(float));
```

See Tensor Data Interface for more details.

57.5.5.57 starpu_tensor_ptr_register()

```
void starpu_tensor_ptr_register (
            starpu_data_handle_t handle,
            unsigned node,
            uintptr_t ptr,
            uintptr_t dev_handle,
            size_t offset,
            uint32_t ldy,
            uint32_t ldz,
            uint32_t ldt )
```

Register into the `handle` that to store data on node `node` it should use the buffer located at `ptr`, or device handle `dev_handle` and offset `offset` (for OpenCL, notably), with `ldy` elements between rows, and `ldz` elements between z planes, and `ldt` elements between t cubes.

57.5.5.58 starpu_tensor_get_nx()

```
uint32_t starpu_tensor_get_nx (
            starpu_data_handle_t handle )
```

Return the number of elements on the x-axis of the tensor designated by `handle`.

57.5.5.59 starpu_tensor_get_ny()

```
uint32_t starpu_tensor_get_ny (
            starpu_data_handle_t handle )
```

Return the number of elements on the y-axis of the tensor designated by `handle`.

57.5.5.60 starpu_tensor_get_nz()

```
uint32_t starpu_tensor_get_nz (
            starpu_data_handle_t handle )
```

Return the number of elements on the z-axis of the tensor designated by `handle`.

57.5.5.61 starpu_tensor_get_nt()

```
uint32_t starpu_tensor_get_nt (
            starpu_data_handle_t handle )
```

Return the number of elements on the t-axis of the tensor designated by `handle`.

57.5.5.62 starpu_tensor_get_local_ldy()

```
uint32_t starpu_tensor_get_local_ldy (
            starpu_data_handle_t handle )
```

Return the number of elements between each row of the tensor designated by `handle`, in the format of the current memory node.

57.5.5.63 starpu_tensor_get_local_ldz()

```
uint32_t starpu_tensor_get_local_ldz (
            starpu_data_handle_t handle )
```

Return the number of elements between each z plane of the tensor designated by `handle`, in the format of the current memory node.

57.5.5.64 starpu_tensor_get_local_ldt()

```
uint32_t starpu_tensor_get_local_ldt (
            starpu_data_handle_t handle )
```

Return the number of elements between each t cubes of the tensor designated by `handle`, in the format of the current memory node.

57.5.5.65 starpu_tensor_get_local_ptr()

```
uintptr_t starpu_tensor_get_local_ptr (
            starpu_data_handle_t handle )
```

Return the local pointer associated with `handle`.

57.5.5.66 starpu_tensor_get_elemsize()

```
size_t starpu_tensor_get_elemsize (
            starpu_data_handle_t handle )
```

Return the size of the elements of the tensor designated by `handle`.

57.5.5.67 starpu_ndim_data_register()

```
void starpu_ndim_data_register (
            starpu_data_handle_t * handleptr,
            int home_node,
            uintptr_t ptr,
            uint32_t * ldn,
            uint32_t * nn,
            size_t ndim,
            size_t elemsize )
```

Register the `nn`[0] x `nn`[1] x ... `ndim-dimension` matrix of `elemsize` byte elements pointed by `ptr` and initialize `handle` to represent it. Again, `ldn`, specifies the number of elements between two units on each dimension.

Here an example of how to use the function.

```
float *ndim_arr;
size_t arrsize = 1;
     int i;
     for (i = 0; i < ndim; i++)
        arrsize = arrsize * nn[i];
starpu_data_handle_t ndim_handle;
ndim_arr = (float*)malloc(arrsize*sizeof(float));
starpu_ndim_data_register(&ndim_handle, STARPU_MAIN_RAM, (uintptr_t)ndim_arr, ldn, nn, ndim, sizeof(float));
```

See Ndim Data Interface for more details.

57.5.5.68 starpu_ndim_ptr_register()

```
void starpu_ndim_ptr_register (
            starpu_data_handle_t handle,
            unsigned node,
            uintptr_t ptr,
            uintptr_t dev_handle,
            size_t offset,
            uint32_t * ldn )
```

Register into the `handle` that to store data on node `node` it should use the buffer located at `ptr`, or device handle `dev_handle` and offset `offset` (for OpenCL, notably), with `ldn` elements between two units on each dimension.

57.5.5.69 starpu_ndim_get_nn()

```
uint32_t * starpu_ndim_get_nn (
            starpu_data_handle_t handle )
```

Return the number of elements on each dimension of the ndim array designated by `handle`.

57.5.5.70 starpu_ndim_get_ni()

```
uint32_t starpu_ndim_get_ni (
            starpu_data_handle_t handle,
            size_t i )
```

Return the number of elements on the i-axis of the ndim array designated by `handle`. When i=0, it means x-axis, when i=1, it means y-axis, when i=2, it means z-axis, etc.

57.5.5.71 starpu_ndim_get_local_ldn()

```
uint32_t * starpu_ndim_get_local_ldn (
            starpu_data_handle_t handle )
```

Return the number of elements between two units on each dimension of the ndim array designated by `handle`, in the format of the current memory node.

57.5.5.72 starpu_ndim_get_local_ldi()

```
uint32_t starpu_ndim_get_local_ldi (
            starpu_data_handle_t handle,
            size_t i )
```

Return the number of elements between two units i-axis dimension of the ndim array designated by `handle`, in the format of the current memory node.

57.5.5.73 starpu_ndim_get_local_ptr()

```
uintptr_t starpu_ndim_get_local_ptr (
            starpu_data_handle_t handle )
```

Return the local pointer associated with `handle`.

57.5.5.74 starpu_ndim_get_ndim()

```
size_t starpu_ndim_get_ndim (
            starpu_data_handle_t handle )
```

Return the dimension size.

57.5.5.75 starpu_ndim_get_elemsize()

```
size_t starpu_ndim_get_elemsize (
            starpu_data_handle_t handle )
```

Return the size of the elements of the ndim array designated by `handle`.

57.5.5.76 starpu_vector_data_register()

```
void starpu_vector_data_register (
            starpu_data_handle_t * handle,
            int home_node,
            uintptr_t ptr,
            uint32_t nx,
            size_t elemsize )
```

Register the `nx` `elemsize-byte` elements pointed to by `ptr` and initialize `handle` to represent it. Here an example of how to use the function.

```
float vector[NX];
starpu_data_handle_t vector_handle;
starpu_vector_data_register(&vector_handle, STARPU_MAIN_RAM, (uintptr_t)vector, NX, sizeof(vector[0]));
```

See Vector Data Interface for more details.

57.5.5.77 starpu_vector_data_register_allocsize()

```
void starpu_vector_data_register_allocsize (
            starpu_data_handle_t * handle,
            int home_node,
            uintptr_t ptr,
            uint32_t nx,
            size_t elemsize,
            size_t allocsize )
```

Similar to starpu_vector_data_register, but additionally specifies which allocation size should be used instead of the initial nx∗elemsize. See Data Interface with Variable Size for more details.

57.5.5.78 starpu_vector_ptr_register()

```
void starpu_vector_ptr_register (
            starpu_data_handle_t handle,
            unsigned node,
            uintptr_t ptr,
            uintptr_t dev_handle,
            size_t offset )
```

Register into the `handle` that to store data on node `node` it should use the buffer located at `ptr`, or device handle `dev_handle` and offset `offset` (for OpenCL, notably)

57.5.5.79 starpu_vector_get_nx()

```
uint32_t starpu_vector_get_nx (
                starpu_data_handle_t handle )
```

Return the number of elements registered into the array designated by `handle`.

57.5.5.80 starpu_vector_get_elemsize()

```
size_t starpu_vector_get_elemsize (
                starpu_data_handle_t handle )
```

Return the size of each element of the array designated by `handle`.

57.5.5.81 starpu_vector_get_allocsize()

```
size_t starpu_vector_get_allocsize (
                starpu_data_handle_t handle )
```

Return the allocated size of the array designated by `handle`.

57.5.5.82 starpu_vector_get_local_ptr()

```
uintptr_t starpu_vector_get_local_ptr (
                starpu_data_handle_t handle )
```

Return the local pointer associated with `handle`.

57.5.5.83 starpu_variable_data_register()

```
void starpu_variable_data_register (
                starpu_data_handle_t * handle,
                int home_node,
                uintptr_t ptr,
                size_t size )
```

Register the `size` byte element pointed to by `ptr`, which is typically a scalar, and initialize `handle` to represent this data item.

Here an example of how to use the function.

```
float var = 42.0;
starpu_data_handle_t var_handle;
starpu_variable_data_register(&var_handle, STARPU_MAIN_RAM, (uintptr_t)&var, sizeof(var));
```

See Variable Data Interface for more details.

57.5.5.84 starpu_variable_ptr_register()

```
void starpu_variable_ptr_register (
                starpu_data_handle_t handle,
                unsigned node,
                uintptr_t ptr,
                uintptr_t dev_handle,
                size_t offset )
```

Register into the `handle` that to store data on node `node` it should use the buffer located at `ptr`, or device handle `dev_handle` and offset `offset` (for OpenCL, notably)

57.5.5.85 starpu_variable_get_elemsize()

```
size_t starpu_variable_get_elemsize (
                starpu_data_handle_t handle )
```

Return the size of the variable designated by `handle`.

57.5.5.86 starpu_variable_get_local_ptr()

```
uintptr_t starpu_variable_get_local_ptr (
             starpu_data_handle_t handle )
```

Return a pointer to the variable designated by `handle`.

57.5.5.87 starpu_void_data_register()

```
void starpu_void_data_register (
             starpu_data_handle_t * handle )
```

Register a void interface. There is no data really associated to that interface, but it may be used as a synchronization mechanism. It also permits to express an abstract piece of data that is managed by the application internally: this makes it possible to forbid the concurrent execution of different tasks accessing the same `void` data in read-write concurrently. See Data handles helpers for more details.

57.5.5.88 starpu_csr_data_register()

```
void starpu_csr_data_register (
             starpu_data_handle_t * handle,
             int home_node,
             uint32_t nnz,
             uint32_t nrow,
             uintptr_t nzval,
             uint32_t * colind,
             uint32_t * rowptr,
             uint32_t firstentry,
             size_t elemsize )
```

Register a CSR (Compressed Sparse Row Representation) sparse matrix. See CSR Data Interface for more details.

57.5.5.89 starpu_csr_get_nnz()

```
uint32_t starpu_csr_get_nnz (
             starpu_data_handle_t handle )
```

Return the number of non-zero values in the matrix designated by `handle`.

57.5.5.90 starpu_csr_get_nrow()

```
uint32_t starpu_csr_get_nrow (
             starpu_data_handle_t handle )
```

Return the size of the row pointer array of the matrix designated by `handle`.

57.5.5.91 starpu_csr_get_firstentry()

```
uint32_t starpu_csr_get_firstentry (
             starpu_data_handle_t handle )
```

Return the index at which all arrays (the column indexes, the row pointers...) of the matrix designated by `handle`.

57.5.5.92 starpu_csr_get_local_nzval()

```
uintptr_t starpu_csr_get_local_nzval (
             starpu_data_handle_t handle )
```

Return a local pointer to the non-zero values of the matrix designated by `handle`.

57.5.5.93 starpu_csr_get_local_colind()

```
uint32_t * starpu_csr_get_local_colind (
            starpu_data_handle_t handle )
```

Return a local pointer to the column index of the matrix designated by `handle`.

57.5.5.94 starpu_csr_get_local_rowptr()

```
uint32_t * starpu_csr_get_local_rowptr (
            starpu_data_handle_t handle )
```

Return a local pointer to the row pointer array of the matrix designated by `handle`.

57.5.5.95 starpu_csr_get_elemsize()

```
size_t starpu_csr_get_elemsize (
            starpu_data_handle_t handle )
```

Return the size of the elements registered into the matrix designated by `handle`.

57.5.5.96 starpu_bcsr_data_register()

```
void starpu_bcsr_data_register (
            starpu_data_handle_t * handle,
            int home_node,
            uint32_t nnz,
            uint32_t nrow,
            uintptr_t nzval,
            uint32_t * colind,
            uint32_t * rowptr,
            uint32_t firstentry,
            uint32_t r,
            uint32_t c,
            size_t elemsize )
```

This variant of starpu_data_register() uses the BCSR (Blocked Compressed Sparse Row Representation) sparse matrix interface. Register the sparse matrix made of `nnz` non-zero blocks of elements of size `elemsize` stored in `nzval` and initializes `handle` to represent it. Blocks have size `r * c`. `nrow` is the number of rows (in terms of blocks), `colind` is an array of nnz elements, colind[i] is the block-column index for block i in `nzval`, `rowptr` is an array of nrow+1 elements, rowptr[i] is the block-index (in `nzval`) of the first block of row i. By convention, rowptr[nrow] is the number of blocks, this allows an easier access of the matrix's elements for the kernels. `firstentry` is the index of the first entry of the given arrays (usually 0 or 1).

Here an example with the following matrix:

```
|  0   1   0   0 |
|  2   3   0   0 |
|  4   5   8   9 |
|  6   7  10  11 |
nzval  = [0, 1, 2, 3] ++ [4, 5, 6, 7] ++ [8, 9, 10, 11]
colind = [0, 0, 1]
rowptr = [0, 1, 3]
r = c = 2
```

which translates into the following code

```
int R = 2; // Size of the blocks
int C = 2;
int NROWS = 2;
int NNZ_BLOCKS = 3;    // out of 4
int NZVAL_SIZE = (R*C*NNZ_BLOCKS);
int nzval[NZVAL_SIZE]  =
{
  0, 1, 2, 3,    // First block
  4, 5, 6, 7,    // Second block
  8, 9, 10, 11   // Third block
};
uint32_t colind[NNZ_BLOCKS] =
{
  0, // block-column index for first block in nzval
  0, // block-column index for second block in nzval
  1  // block-column index for third block in nzval
};
uint32_t rowptr[NROWS+1] =
{
  0, // block-index in nzval of the first block of the first row.
  1, // block-index in nzval of the first block of the second row.
  NNZ_BLOCKS // number of blocks, to allow an easier element's access for the kernels
};
starpu_data_handle_t bcsr_handle;
starpu_bcsr_data_register(&bcsr_handle,
                   STARPU_MAIN_RAM,
                   NNZ_BLOCKS,
                   NROWS,
                   (uintptr_t) nzval,
                   colind,
                   rowptr,
                   0, // firstentry
                   R,
                   C,
                   sizeof(nzval[0]));
```

See BCSR Data Interface for more details.

57.5.5.97 starpu_bcsr_get_nnz()

```
uint32_t starpu_bcsr_get_nnz (
            starpu_data_handle_t handle )
```

Return the number of non-zero elements in the matrix designated by `handle`.

57.5.5.98 starpu_bcsr_get_nrow()

```
uint32_t starpu_bcsr_get_nrow (
            starpu_data_handle_t handle )
```

Return the number of rows (in terms of blocks of size r∗c) in the matrix designated by `handle`.

57.5.5.99 starpu_bcsr_get_firstentry()

```
uint32_t starpu_bcsr_get_firstentry (
            starpu_data_handle_t handle )
```

Return the index at which all arrays (the column indexes, the row pointers...) of the matrix desginated by `handle`.

57.5.5.100 starpu_bcsr_get_local_nzval()

```
uintptr_t starpu_bcsr_get_local_nzval (
            starpu_data_handle_t handle )
```

Return a pointer to the non-zero values of the matrix designated by `handle`.

57.5.5.101 starpu_bcsr_get_local_colind()

```
uint32_t * starpu_bcsr_get_local_colind (
            starpu_data_handle_t handle )
```

Return a pointer to the column index, which holds the positions of the non-zero entries in the matrix designated by `handle`.

57.5.5.102 starpu_bcsr_get_local_rowptr()

```
uint32_t * starpu_bcsr_get_local_rowptr (
            starpu_data_handle_t handle )
```

Return the row pointer array of the matrix designated by `handle`.

57.5.5.103 starpu_bcsr_get_r()

```
uint32_t starpu_bcsr_get_r (
            starpu_data_handle_t handle )
```

Return the number of rows in a block.

57.5.5.104 starpu_bcsr_get_c()

```
uint32_t starpu_bcsr_get_c (
            starpu_data_handle_t handle )
```

Return the number of columns in a block.

57.5.5.105 starpu_bcsr_get_elemsize()

```
size_t starpu_bcsr_get_elemsize (
            starpu_data_handle_t handle )
```

Return the size of the elements in the matrix designated by `handle`.

57.5.5.106 starpu_multiformat_data_register()

```
void starpu_multiformat_data_register (
            starpu_data_handle_t * handle,
            int home_node,
            void * ptr,
            uint32_t nobjects,
            struct starpu_multiformat_data_interface_ops * format_ops )
```

Register a piece of data that can be represented in different ways, depending upon the processing unit that manipulates it. It allows the programmer, for instance, to use an array of structures when working on a CPU, and a structure of arrays when working on a GPU. `nobjects` is the number of elements in the data. `format_ops` describes the format. See The Multiformat Interface for more details.

57.5.5.107 starpu_hash_crc32c_be_n()

```
uint32_t starpu_hash_crc32c_be_n (
            const void * input,
            size_t n,
            uint32_t inputcrc )
```

Compute the CRC of a byte buffer seeded by the `inputcrc` *current state*. The return value should be considered as the new *current state* for future CRC computation. This is used for computing data size footprint. See Data footprint for more details.

57.5.5.108 starpu_hash_crc32c_be_ptr()

```
uint32_t starpu_hash_crc32c_be_ptr (
            void * input,
            uint32_t inputcrc )
```

Compute the CRC of a pointer value seeded by the `inputcrc` *current state*. The return value should be considered as the new *current state* for future CRC computation. This is used for computing data size footprint. See Data footprint for more details.

57.5.5.109 starpu_hash_crc32c_be()

```
uint32_t starpu_hash_crc32c_be (
            uint32_t input,
            uint32_t inputcrc )
```

Compute the CRC of a 32bit number seeded by the `inputcrc` *current state*. The return value should be considered as the new *current state* for future CRC computation. This is used for computing data size footprint. See Data footprint for more details.

57.5.5.110 starpu_hash_crc32c_string()

```
uint32_t starpu_hash_crc32c_string (
            const char * str,
            uint32_t inputcrc )
```

Compute the CRC of a string seeded by the `inputcrc` *current state*. The return value should be considered as the new *current state* for future CRC computation. This is used for computing data size footprint. See Data footprint for more details.

57.6 Data Management

Data management facilities provided by StarPU. We show how to use existing data interfaces in Data Interfaces, but developers can design their own data interfaces if required.

Typedefs

- typedef struct _starpu_data_state ∗ starpu_data_handle_t
- typedef struct starpu_arbiter ∗ starpu_arbiter_t

Enumerations

- enum starpu_data_access_mode {
  STARPU_NONE , STARPU_R , STARPU_W , STARPU_RW ,
  STARPU_SCRATCH , STARPU_REDUX , STARPU_COMMUTE , STARPU_SSEND ,
  STARPU_LOCALITY , STARPU_MPI_REDUX , STARPU_NOPLAN , STARPU_UNMAP ,
  STARPU_NOFOOTPRINT , STARPU_ACCESS_MODE_MAX }
- enum starpu_is_prefetch {
  STARPU_FETCH , STARPU_TASK_PREFETCH , STARPU_PREFETCH , STARPU_IDLEFETCH ,
  STARPU_NFETCH }

Functions

- void starpu_data_set_name (starpu_data_handle_t handle, const char ∗name)
- void starpu_data_set_coordinates_array (starpu_data_handle_t handle, unsigned dimensions, int dims[ ])
- void starpu_data_set_coordinates (starpu_data_handle_t handle, unsigned dimensions,...)
- unsigned starpu_data_get_coordinates_array (starpu_data_handle_t handle, unsigned dimensions, int dims[ ])
- void starpu_data_unregister (starpu_data_handle_t handle)
- void starpu_data_unregister_no_coherency (starpu_data_handle_t handle)
- void starpu_data_unregister_submit (starpu_data_handle_t handle)
- void starpu_data_deinitialize (starpu_data_handle_t handle)
- void starpu_data_deinitialize_submit (starpu_data_handle_t handle)
- void starpu_data_invalidate (starpu_data_handle_t handle)
- void starpu_data_invalidate_submit (starpu_data_handle_t handle)
- void starpu_data_advise_as_important (starpu_data_handle_t handle, unsigned is_important)
- starpu_arbiter_t starpu_arbiter_create (void) STARPU_ATTRIBUTE_MALLOC
- void starpu_data_assign_arbiter (starpu_data_handle_t handle, starpu_arbiter_t arbiter)
- void starpu_arbiter_destroy (starpu_arbiter_t arbiter)
- int starpu_data_request_allocation (starpu_data_handle_t handle, unsigned node)
- int starpu_data_fetch_on_node (starpu_data_handle_t handle, unsigned node, unsigned async)
- int starpu_data_prefetch_on_node (starpu_data_handle_t handle, unsigned node, unsigned async)
- int starpu_data_prefetch_on_node_prio (starpu_data_handle_t handle, unsigned node, unsigned async, int prio)
- int starpu_data_idle_prefetch_on_node (starpu_data_handle_t handle, unsigned node, unsigned async)
- int starpu_data_idle_prefetch_on_node_prio (starpu_data_handle_t handle, unsigned node, unsigned async, int prio)
- unsigned starpu_data_is_on_node (starpu_data_handle_t handle, unsigned node)
- void starpu_data_wont_use (starpu_data_handle_t handle)
- int starpu_data_evict_from_node (starpu_data_handle_t handle, unsigned node)
- void starpu_data_set_wt_mask (starpu_data_handle_t handle, uint32_t wt_mask)
- void starpu_data_set_ooc_flag (starpu_data_handle_t handle, unsigned flag)
- unsigned starpu_data_get_ooc_flag (starpu_data_handle_t handle)
- void starpu_data_query_status2 (starpu_data_handle_t handle, int memory_node, int ∗is_allocated, int ∗is_valid, int ∗is_loading, int ∗is_requested)

- void starpu_data_query_status (starpu_data_handle_t handle, int memory_node, int ∗is_allocated, int ∗is_valid, int ∗is_requested)
- void starpu_data_set_reduction_methods (starpu_data_handle_t handle, struct starpu_codelet ∗redux_cl, struct starpu_codelet ∗init_cl)
- void starpu_data_set_reduction_methods_with_args (starpu_data_handle_t handle, struct starpu_codelet ∗redux_cl, void ∗redux_cl_arg, struct starpu_codelet ∗init_cl, void ∗init_cl_arg)
- struct starpu_data_interface_ops ∗ **starpu_data_get_interface_ops** (starpu_data_handle_t handle)
- unsigned starpu_data_test_if_allocated_on_node (starpu_data_handle_t handle, unsigned memory_node)
- unsigned starpu_data_test_if_mapped_on_node (starpu_data_handle_t handle, unsigned memory_node)
- void starpu_memchunk_tidy (unsigned memory_node)
- void starpu_data_set_user_data (starpu_data_handle_t handle, void ∗user_data)
- void ∗ starpu_data_get_user_data (starpu_data_handle_t handle)
- void starpu_data_set_sched_data (starpu_data_handle_t handle, void ∗sched_data)
- void ∗ starpu_data_get_sched_data (starpu_data_handle_t handle)
- int starpu_data_can_evict (starpu_data_handle_t handle, unsigned node, enum starpu_is_prefetch is_prefetch)

Access registered data from the application

- int starpu_data_acquire (starpu_data_handle_t handle, enum starpu_data_access_mode mode)
- int starpu_data_acquire_on_node (starpu_data_handle_t handle, int node, enum starpu_data_access_mode mode)
- int starpu_data_acquire_cb (starpu_data_handle_t handle, enum starpu_data_access_mode mode, void(∗callback)(void ∗), void ∗arg)
- int starpu_data_acquire_on_node_cb (starpu_data_handle_t handle, int node, enum starpu_data_access_mode mode, void(∗callback)(void ∗), void ∗arg)
- int starpu_data_acquire_cb_sequential_consistency (starpu_data_handle_t handle, enum starpu_data_access_mode mode, void(∗callback)(void ∗), void ∗arg, int sequential_consistency)
- int starpu_data_acquire_on_node_cb_sequential_consistency (starpu_data_handle_t handle, int node, enum starpu_data_access_mode mode, void(∗callback)(void ∗), void ∗arg, int sequential_consistency)
- int starpu_data_acquire_on_node_cb_sequential_consistency_sync_jobids (starpu_data_handle_t handle, int node, enum starpu_data_access_mode mode, void(∗callback_acquired)(void ∗arg, int ∗node, enum starpu_data_access_mode mode), void(∗callback)(void ∗arg), void ∗arg, int sequential_consistency, int quick, long ∗pre_sync_jobid, long ∗post_sync_jobid, int prio)
- int starpu_data_acquire_try (starpu_data_handle_t handle, enum starpu_data_access_mode mode)
- int starpu_data_acquire_on_node_try (starpu_data_handle_t handle, int node, enum starpu_data_access_mode mode)
- void starpu_data_release (starpu_data_handle_t handle)
- void starpu_data_release_on_node (starpu_data_handle_t handle, int node)
- void starpu_data_release_to (starpu_data_handle_t handle, enum starpu_data_access_mode down_to_mode)
- void starpu_data_release_to_on_node (starpu_data_handle_t handle, enum starpu_data_access_mode down_to_mode, int node)
- #define STARPU_ACQUIRE_NO_NODE
- #define STARPU_ACQUIRE_NO_NODE_LOCK_ALL
- #define STARPU_DATA_ACQUIRE_CB(handle, mode, code)

Implicit Data Dependencies

In this section, we describe how StarPU makes it possible to insert implicit task dependencies in order to enforce sequential data consistency. When this data consistency is enabled on a specific data handle, any data access will appear as sequentially consistent from the application. For instance, if the application submits two tasks that access the same piece of data in read-only mode, and then a third task that access it in write mode, dependencies will be added between the two first tasks and the third one. Implicit data dependencies are also inserted in the case of data accesses from the application.

- void starpu_data_set_sequential_consistency_flag (starpu_data_handle_t handle, unsigned flag)

- unsigned starpu_data_get_sequential_consistency_flag (starpu_data_handle_t handle)
- unsigned starpu_data_get_default_sequential_consistency_flag (void)
- void starpu_data_set_default_sequential_consistency_flag (unsigned flag)

57.6.1 Detailed Description

Data management facilities provided by StarPU. We show how to use existing data interfaces in Data Interfaces, but developers can design their own data interfaces if required.

57.6.2 Macro Definition Documentation

57.6.2.1 STARPU_ACQUIRE_NO_NODE

`#define STARPU_ACQUIRE_NO_NODE`

This macro can be used to acquire data, but not require it to be available on a given node, only enforce R/W dependencies. This can for instance be used to wait for tasks which produce the data, but without requesting a fetch to the main memory.

57.6.2.2 STARPU_ACQUIRE_NO_NODE_LOCK_ALL

`#define STARPU_ACQUIRE_NO_NODE_LOCK_ALL`

Similar to STARPU_ACQUIRE_NO_NODE, but will lock the data on all nodes, preventing them from being evicted for instance. This is mostly useful inside StarPU only.

57.6.2.3 STARPU_DATA_ACQUIRE_CB

```
#define STARPU_DATA_ACQUIRE_CB(
                handle,
                mode,
                code )
```

STARPU_DATA_ACQUIRE_CB() is the same as starpu_data_acquire_cb(), except that the code to be executed in a callback is directly provided as a macro parameter, and the data `handle` is automatically released after it. This permits to easily execute code which depends on the value of some registered data. This is non-blocking too and may be called from task callbacks.

57.6.3 Typedef Documentation

57.6.3.1 starpu_data_handle_t

`typedef struct _starpu_data_state* starpu_data_handle_t`

StarPU uses starpu_data_handle_t as an opaque handle to manage a piece of data. Once a piece of data has been registered to StarPU, it is associated to a starpu_data_handle_t which keeps track of the state of the piece of data over the entire machine, so that we can maintain data consistency and locate data replicates for instance. See Data Interface for more details.

57.6.3.2 starpu_arbiter_t

`typedef struct starpu_arbiter* starpu_arbiter_t`

This is an arbiter, which implements an advanced but centralized management of concurrent data accesses, see Concurrent Data Accesses for the details.

57.6.4 Enumeration Type Documentation

57.6.4.1 starpu_data_access_mode

`enum starpu_data_access_mode`

Describe a StarPU data access mode
Note: when adding a flag here, update _starpu_detect_implicit_data_deps_with_handle
Note: other STARPU_∗ values in include/starpu_task_util.h

Enumerator

STARPU_NONE	todo
STARPU_R	read-only mode
STARPU_W	write-only mode
STARPU_RW	read-write mode. Equivalent to STARPU_R|STARPU_W
STARPU_SCRATCH	A temporary buffer is allocated for the task, but StarPU does not enforce data consistency—i.e. each device has its own buffer, independently from each other (even for CPUs), and no data transfer is ever performed. This is useful for temporary variables to avoid allocating/freeing buffers inside each task. Currently, no behavior is defined concerning the relation with the STARPU_R and STARPU_W modes and the value provided at registration — i.e., the value of the scratch buffer is undefined at entry of the codelet function. It is being considered for future extensions at least to define the initial value. For now, data to be used in STARPU_SCRATCH mode should be registered with node -1 and a `NULL` pointer, since the value of the provided buffer is simply ignored for now. See Scratch Data for more details.
STARPU_REDUX	Reduction mode. StarPU will allocate on the fly a per-worker buffer, so that various tasks that access the same data in STARPU_REDUX mode can execute in parallel. When a task accesses the data without STARPU_REDUX, StarPU will automatically reduce the different contributions. Codelets contributing to these reductions with STARPU_REDUX must be registered with STARPU_RW | STARPU_COMMUTE access modes. See Data Reduction for more details.
STARPU_COMMUTE	STARPU_COMMUTE can be passed along STARPU_W or STARPU_RW to express that StarPU can let tasks commute, which is useful e.g. when bringing a contribution into some data, which can be done in any order (but still require sequential consistency against reads or non-commutative writes). See Commute Data Access for more details.
STARPU_SSEND	used in starpu_mpi_task_insert() to specify the data has to be sent using a synchronous and non-blocking mode (see starpu_mpi_issend())
STARPU_LOCALITY	used to tell the scheduler which data is the most important for the task, and should thus be used to try to group tasks on the same core or cache, etc. For now only the ws and lws schedulers take this flag into account, and only when rebuild with `USE_LOCALITY` flag defined in the src/sched_policies/work_stealing_policy.c source code. TODO add extended description in documentation.
STARPU_MPI_REDUX	Inter-node reduction only. This is similar to STARPU_REDUX, except that StarPU will allocate a per-node buffer only, i.e. parallelism will be achieved between nodes, but not within each node. This is useful when the per-worker buffers allocated with STARPU_REDUX consume too much memory. See Inter-node reduction for more details.
STARPU_NOPLAN	Disable automatic submission of asynchronous partitioning/unpartitioning, only use internally by StarPU
STARPU_UNMAP	Request unmapping the destination replicate, only use internally by StarPU

Enumerator

STARPU_NOFOOTPRINT	Ignore this data for the footprint computation. See Scratch Data
STARPU_ACCESS_MODE_MAX	The purpose of STARPU_ACCESS_MODE_MAX is to be the maximum of this enum.

57.6.4.2 starpu_is_prefetch

```
enum starpu_is_prefetch
```

Prefetch levels
Data requests are ordered by priorities, but also by prefetching level, between data that a task wants now, and data that we will probably want "soon".

Enumerator

STARPU_FETCH	A task really needs it now!
STARPU_TASK_PREFETCH	A task will need it soon
STARPU_PREFETCH	It is a good idea to have it asap
STARPU_IDLEFETCH	Get this here when you have time to

57.6.5 Function Documentation

57.6.5.1 starpu_data_set_name()

```
void starpu_data_set_name (
            starpu_data_handle_t handle,
            const char * name )
```

Set the name of the data, to be shown in various profiling tools. See Creating a Gantt Diagram for more details.

57.6.5.2 starpu_data_set_coordinates_array()

```
void starpu_data_set_coordinates_array (
            starpu_data_handle_t handle,
            unsigned dimensions,
            int dims[ ] )
```

Set the coordinates of the data, to be shown in various profiling tools. `dimensions` is the size of the `dims` array. This can be for instance the tile coordinates within a big matrix. See Creating a Gantt Diagram for more details.

57.6.5.3 starpu_data_set_coordinates()

```
void starpu_data_set_coordinates (
            starpu_data_handle_t handle,
            unsigned dimensions,
             ... )
```

Set the coordinates of the data, to be shown in various profiling tools. `dimensions` is the number of subsequent `int` parameters. This can be for instance the tile coordinates within a big matrix. See Creating a Gantt Diagram for more details.

57.6.5.4 starpu_data_get_coordinates_array()

```
unsigned starpu_data_get_coordinates_array (
            starpu_data_handle_t handle,
            unsigned dimensions,
            int dims[ ] )
```

Get the coordinates of the data, as set by a previous call to starpu_data_set_coordinates_array() or starpu_data_set_coordinates() `dimensions` is the size of the `dims` array. This returns the actual number of returned coordinates. See Creating a Gantt Diagram for more details.

57.6.5.5 starpu_data_unregister()

```
void starpu_data_unregister (
            starpu_data_handle_t handle )
```

Unregister a data `handle` from StarPU. If the data was automatically allocated by StarPU because the home node was -1, all automatically allocated buffers are freed. Otherwise, a valid copy of the data is put back into the home node in the buffer that was initially registered. Using a data handle that has been unregistered from StarPU results in an undefined behaviour. In case we do not need to update the value of the data in the home node, we can use the function starpu_data_unregister_no_coherency() instead. See Task Submission for more details.

57.6.5.6 starpu_data_unregister_no_coherency()

```
void starpu_data_unregister_no_coherency (
            starpu_data_handle_t handle )
```

Similar to starpu_data_unregister(), except that StarPU does not put back a valid copy into the home node, in the buffer that was initially registered. See Data Management Allocation for more details.

57.6.5.7 starpu_data_unregister_submit()

```
void starpu_data_unregister_submit (
            starpu_data_handle_t handle )
```

Destroy the data `handle` once it is no longer needed by any submitted task. No coherency is provided.
This is not safe to call starpu_data_unregister_submit() on a handle that comes from the registration of a non-NULL application home buffer, since the moment when the unregistration will happen is unknown to the application. Only calling starpu_shutdown() allows to be sure that the data was really unregistered. See Temporary Data for more details.

57.6.5.8 starpu_data_deinitialize()

```
void starpu_data_deinitialize (
            starpu_data_handle_t handle )
```

Deinitialize all replicates of the data `handle` immediately. After data deinitialization, the first access to `handle` must be performed in STARPU_W mode. Accessing an deinitialized data in STARPU_R mode results in undefined behaviour. See Data Management Allocation for more details.

57.6.5.9 starpu_data_deinitialize_submit()

```
void starpu_data_deinitialize_submit (
            starpu_data_handle_t handle )
```

Submit deinitialization of the data `handle` after completion of previously submitted tasks. See Data Management Allocation for more details.

57.6.5.10 starpu_data_invalidate()

```
void starpu_data_invalidate (
            starpu_data_handle_t handle )
```

Destroy all replicates of the data `handle` immediately. After data invalidation, the first access to `handle` must be performed in STARPU_W mode. Accessing an invalidated data in STARPU_R mode results in undefined behaviour. See Data Management Allocation for more details.
This is the same as starpu_data_deinitialize(), plus explicitly releasing the buffers.

57.6.5.11 starpu_data_invalidate_submit()

```
void starpu_data_invalidate_submit (
            starpu_data_handle_t handle )
```

Submit invalidation of the data `handle` after completion of previously submitted tasks. See Data Management Allocation for more details.
This is the same as starpu_data_deinitialize_submit(), plus explicitly releasing the buffers.

57.6.5.12 starpu_data_advise_as_important()

```
void starpu_data_advise_as_important (
            starpu_data_handle_t handle,
            unsigned is_important )
```

Specify that the data `handle` can be discarded without impacting the application.

57.6.5.13 starpu_data_acquire()

```
int starpu_data_acquire (
            starpu_data_handle_t handle,
            enum starpu_data_access_mode mode )
```

The application must call this function prior to accessing registered data from main memory outside tasks. StarPU ensures that the application will get an up-to-date copy of `handle` in main memory located where the data was originally registered, and that all concurrent accesses (e.g. from tasks) will be consistent with the access mode specified with `mode`. starpu_data_release() must be called once the application no longer needs to access the piece of data. Note that implicit data dependencies are also enforced by starpu_data_acquire(), i.e. starpu_data_acquire() will wait for all tasks scheduled to work on the data, unless they have been disabled explicitly by calling starpu_data_set_default_sequential_consistency_flag() or starpu_data_set_sequential_consistency_flag(). starpu_data_acquire() is a blocking call, so that it cannot be called from tasks or from their callbacks (in that case, starpu_data_acquire() returns `-EDEADLK`). Upon successful completion, this function returns 0. See Data Access for more details.

57.6.5.14 starpu_data_acquire_on_node()

```
int starpu_data_acquire_on_node (
            starpu_data_handle_t handle,
            int node,
            enum starpu_data_access_mode mode )
```

Similar to starpu_data_acquire(), except that the data will be available on the given memory node instead of main memory. STARPU_ACQUIRE_NO_NODE and STARPU_ACQUIRE_NO_NODE_LOCK_ALL can be used instead of an explicit node number. See Data Access for more details.

57.6.5.15 starpu_data_acquire_cb()

```
int starpu_data_acquire_cb (
            starpu_data_handle_t handle,
            enum starpu_data_access_mode mode,
            void(*)(void *) callback,
            void * arg )
```

Asynchronous equivalent of starpu_data_acquire(). When the data specified in `handle` is available in the access `mode`, the `callback` function is executed. The application may access the requested data during the execution of `callback`. The `callback` function must call starpu_data_release() once the application no longer needs to access the piece of data. Note that implicit data dependencies are also enforced by starpu_data_acquire_cb() in case they are not disabled. Contrary to starpu_data_acquire(), this function is non-blocking and may be called from task callbacks. Upon successful completion, this function returns 0. See Data Access for more details.

57.6.5.16 starpu_data_acquire_on_node_cb()

```
int starpu_data_acquire_on_node_cb (
            starpu_data_handle_t handle,
            int node,
            enum starpu_data_access_mode mode,
            void(*)(void *) callback,
            void * arg )
```

Similar to starpu_data_acquire_cb(), except that the data will be available on the given memory node instead of main memory. STARPU_ACQUIRE_NO_NODE and STARPU_ACQUIRE_NO_NODE_LOCK_ALL can be used instead of an explicit node number. See Data Access for more details.

57.6.5.17 starpu_data_acquire_cb_sequential_consistency()

```
int starpu_data_acquire_cb_sequential_consistency (
            starpu_data_handle_t handle,
            enum starpu_data_access_mode mode,
            void(*)(void *) callback,
            void * arg,
            int sequential_consistency )
```

Similar to starpu_data_acquire_cb() with the possibility of enabling or disabling data dependencies. When the data specified in `handle` is available in the access `mode`, the `callback` function is executed. The application may access the requested data during the execution of this `callback`. The `callback` function must call starpu_data_release() once the application no longer needs to access the piece of data. Note that implicit data dependencies are also enforced by starpu_data_acquire_cb_sequential_consistency() in case they are not disabled specifically for the given `handle` or by the parameter `sequential_consistency`. Similarly to starpu_data_acquire_cb(), this function is non-blocking and may be called from task callbacks. Upon successful completion, this function returns 0. See Data Access for more details.

57.6.5.18 starpu_data_acquire_on_node_cb_sequential_consistency()

```
int starpu_data_acquire_on_node_cb_sequential_consistency (
            starpu_data_handle_t handle,
            int node,
            enum starpu_data_access_mode mode,
            void(*)(void *) callback,
            void * arg,
            int sequential_consistency )
```

Similar to starpu_data_acquire_cb_sequential_consistency(), except that the data will be available on the given memory node instead of main memory. STARPU_ACQUIRE_NO_NODE and STARPU_ACQUIRE_NO_NODE_LOCK_ALL can be used instead of an explicit node number. See Data Access for more details.

57.6.5.19 starpu_data_acquire_on_node_cb_sequential_consistency_sync_jobids()

```
int starpu_data_acquire_on_node_cb_sequential_consistency_sync_jobids (
            starpu_data_handle_t handle,
            int node,
            enum starpu_data_access_mode mode,
            void(*)(void *arg, int *node, enum starpu_data_access_mode mode) callback_acquired,
            void(*)(void *arg) callback,
            void * arg,
            int sequential_consistency,
            int quick,
            long * pre_sync_jobid,
            long * post_sync_jobid,
            int prio )
```

Similar to starpu_data_acquire_on_node_cb_sequential_consistency(), except that the *pre_sync_jobid* and *post_sync_jobid* parameters can be used to retrieve the jobid of the synchronization tasks. *pre_sync_jobid* happens just before the acquisition, and *post_sync_jobid* happens just after the release.

`callback_acquired` is called when the data is acquired in terms of semantic, but the data is not fetched yet. It is given a pointer to the node, which it can modify if it wishes so.

This is a very internal interface, subject to changes, do not use this.

57.6.5.20 starpu_data_acquire_try()

```
int starpu_data_acquire_try (
            starpu_data_handle_t handle,
            enum starpu_data_access_mode mode )
```

The application can call this function instead of starpu_data_acquire() so as to acquire the data like starpu_data_acquire(), but only if all previously-submitted tasks have completed, in which case starpu_data_acquire_try() returns 0. StarPU will have ensured that the application will get an up-to-date copy of `handle` in main memory located where the data was originally registered. starpu_data_release() must be called once the application no longer needs to access the piece of data. See Data Access for more details.

57.6.5.21 starpu_data_acquire_on_node_try()

```
int starpu_data_acquire_on_node_try (
            starpu_data_handle_t handle,
            int node,
            enum starpu_data_access_mode mode )
```

Similar to starpu_data_acquire_try(), except that the data will be available on the given memory node instead of main memory. STARPU_ACQUIRE_NO_NODE and STARPU_ACQUIRE_NO_NODE_LOCK_ALL can be used instead of an explicit node number. See Data Access for more details.

57.6.5.22 starpu_data_release()

```
void starpu_data_release (
            starpu_data_handle_t handle )
```

Release the piece of data acquired by the application either by starpu_data_acquire() or by starpu_data_acquire_cb(). See Data Access for more details.

57.6.5.23 starpu_data_release_on_node()

```
void starpu_data_release_on_node (
            starpu_data_handle_t handle,
            int node )
```

Similar to starpu_data_release(), except that the data was made available on the given memory `node` instead of main memory. The `node` parameter must be exactly the same as the corresponding `starpu_data_acquire_on_node*` call. See Data Access for more details.

57.6.5.24 starpu_data_release_to()

```
void starpu_data_release_to (
            starpu_data_handle_t handle,
            enum starpu_data_access_mode down_to_mode )
```

Partly release the piece of data acquired by the application either by starpu_data_acquire() or by starpu_data_acquire_cb(), switching the acquisition down to `down_to_mode`. For now, only releasing from STARPU_RW or STARPU_W acquisition down to STARPU_R is supported, or down to the same acquisition. STARPU_NONE can also be passed as `down_to_mode`, in which case this is equivalent to calling starpu_data_release(). See Data Access for more details.

57.6.5.25 starpu_data_release_to_on_node()

```
void starpu_data_release_to_on_node (
            starpu_data_handle_t handle,
            enum starpu_data_access_mode down_to_mode,
            int node )
```

Similar to starpu_data_release_to(), except that the data was made available on the given memory `node` instead of main memory. The `node` parameter must be exactly the same as the corresponding `starpu_data_acquire_on_node*` call. See Data Access for more details.

57.6.5.26 starpu_arbiter_create()

```
starpu_arbiter_t starpu_arbiter_create (
             void  )
```

Create a data access arbiter, see Concurrent Data Accesses for the details

57.6.5.27 starpu_data_assign_arbiter()

```
void starpu_data_assign_arbiter (
             starpu_data_handle_t handle,
             starpu_arbiter_t arbiter )
```

Make access to `handle` managed by `arbiter`, see Concurrent Data Accesses for the details.

57.6.5.28 starpu_arbiter_destroy()

```
void starpu_arbiter_destroy (
             starpu_arbiter_t arbiter )
```

Destroy the `arbiter`. This must only be called after all data assigned to it have been unregistered. See Concurrent Data Accesses for the details.

57.6.5.29 starpu_data_request_allocation()

```
int starpu_data_request_allocation (
             starpu_data_handle_t handle,
             unsigned node )
```

Explicitly ask StarPU to allocate room for a piece of data on the specified memory `node`. See Data Prefetch for more details.

57.6.5.30 starpu_data_fetch_on_node()

```
int starpu_data_fetch_on_node (
             starpu_data_handle_t handle,
             unsigned node,
             unsigned async )
```

Issue a fetch request for the data `handle` to `node`, i.e. requests that the data be replicated to the given node as soon as possible, so that it is available there for tasks. If `async` is 0, the call will block until the transfer is achieved, else the call will return immediately, after having just queued the request. In the latter case, the request will asynchronously wait for the completion of any task writing on the data. See Data Prefetch for more details.

57.6.5.31 starpu_data_prefetch_on_node()

```
int starpu_data_prefetch_on_node (
             starpu_data_handle_t handle,
             unsigned node,
             unsigned async )
```

Issue a prefetch request for the data `handle` to `node`, i.e. requests that the data be replicated to `node` when there is room for it, so that it is available there for tasks. If `async` is 0, the call will block until the transfer is achieved, else the call will return immediately, after having just queued the request. In the latter case, the request will asynchronously wait for the completion of any task writing on the data. See Data Prefetch for more details.

57.6.5.32 starpu_data_prefetch_on_node_prio()

```
int starpu_data_prefetch_on_node_prio (
             starpu_data_handle_t handle,
             unsigned node,
             unsigned async,
             int prio )
```

See Data Prefetch for more details.

57.6.5.33 starpu_data_idle_prefetch_on_node()

```
int starpu_data_idle_prefetch_on_node (
            starpu_data_handle_t handle,
            unsigned node,
            unsigned async )
```

Issue an idle prefetch request for the data `handle` to `node`, i.e. requests that the data be replicated to `node`, so that it is available there for tasks, but only when the bus is really idle. If `async` is 0, the call will block until the transfer is achieved, else the call will return immediately, after having just queued the request. In the latter case, the request will asynchronously wait for the completion of any task writing on the data. See Data Prefetch for more details.

57.6.5.34 starpu_data_idle_prefetch_on_node_prio()

```
int starpu_data_idle_prefetch_on_node_prio (
            starpu_data_handle_t handle,
            unsigned node,
            unsigned async,
            int prio )
```

See Data Prefetch for more details.

57.6.5.35 starpu_data_is_on_node()

```
unsigned starpu_data_is_on_node (
            starpu_data_handle_t handle,
            unsigned node )
```

Check whether a valid copy of `handle` is currently available on memory node `node` (or a transfer request for getting so is ongoing). See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.6.5.36 starpu_data_wont_use()

```
void starpu_data_wont_use (
            starpu_data_handle_t handle )
```

Advise StarPU that `handle` will not be used in the close future, and is thus a good candidate for eviction from GPUs. StarPU will thus write its value back to its home node when the bus is idle, and select this data in priority for eviction when memory gets low. See Data Prefetch for more details.

57.6.5.37 starpu_data_evict_from_node()

```
int starpu_data_evict_from_node (
            starpu_data_handle_t handle,
            unsigned node )
```

Advise StarPU to evict `handle` from the memory node `node` StarPU will thus write its value back to its home node, before evicting it. This may however fail if e.g. some task is still working on it.
If the eviction was successful, 0 is returned ; -1 is returned otherwise.
See Data Prefetch for more details.

57.6.5.38 starpu_data_set_wt_mask()

```
void starpu_data_set_wt_mask (
            starpu_data_handle_t handle,
            uint32_t wt_mask )
```

Set the write-through mask of the data `handle` (and its children), i.e. a bitmask of nodes where the data should be always replicated after modification. It also prevents the data from being evicted from these nodes when memory gets scarce. When the data is modified, it is automatically transferred into those memory nodes. For instance a `1<<0` write-through mask means that the CUDA workers will commit their changes in main memory (node 0). See Data Management Allocation for more details.

57.6.5.39 starpu_data_set_sequential_consistency_flag()

```
void starpu_data_set_sequential_consistency_flag (
            starpu_data_handle_t handle,
            unsigned flag )
```

Set the data consistency mode associated to a data handle. The consistency mode set using this function has the priority over the default mode which can be set with starpu_data_set_default_sequential_consistency_flag(). See Sequential Consistency and Data Management Allocation for more details.

57.6.5.40 starpu_data_get_sequential_consistency_flag()

```
unsigned starpu_data_get_sequential_consistency_flag (
            starpu_data_handle_t handle )
```

Get the data consistency mode associated to the data handle `handle`. See Sequential Consistency for more details.

57.6.5.41 starpu_data_get_default_sequential_consistency_flag()

```
unsigned starpu_data_get_default_sequential_consistency_flag (
            void  )
```

Return the default sequential consistency flag. See Sequential Consistency for more details.

57.6.5.42 starpu_data_set_default_sequential_consistency_flag()

```
void starpu_data_set_default_sequential_consistency_flag (
            unsigned flag )
```

Set the default sequential consistency flag. If a non-zero value is passed, a sequential data consistency will be enforced for all handles registered after this function call, otherwise it is disabled. By default, StarPU enables sequential data consistency. It is also possible to select the data consistency mode of a specific data handle with the function starpu_data_set_sequential_consistency_flag(). See Sequential Consistency for more details.

57.6.5.43 starpu_data_set_ooc_flag()

```
void starpu_data_set_ooc_flag (
            starpu_data_handle_t handle,
            unsigned flag )
```

Set whether this data should be elligible to be evicted to disk storage (1) or not (0). The default is 1. See Data Registration for more details.

57.6.5.44 starpu_data_get_ooc_flag()

```
unsigned starpu_data_get_ooc_flag (
            starpu_data_handle_t handle )
```

Get whether this data was set to be elligible to be evicted to disk storage (1) or not (0). See Data Registration for more details.

57.6.5.45 starpu_data_query_status2()

```
void starpu_data_query_status2 (
            starpu_data_handle_t handle,
            int memory_node,
            int * is_allocated,
            int * is_valid,
            int * is_loading,
            int * is_requested )
```

Query the status of `handle` on the specified `memory_node`.
`is_allocated` tells whether memory was allocated there for the data. `is_valid` tells whether the actual value is available there. `is_loading` tells whether the actual value is getting loaded there. `is_requested` tells whether the actual value is requested to be loaded there by some fetch/prefetch/idlefetch request. See Data Prefetch for more details.

57.6.5.46 starpu_data_query_status()

```
void starpu_data_query_status (
            starpu_data_handle_t handle,
            int memory_node,
            int * is_allocated,
            int * is_valid,
            int * is_requested )
```

Same as starpu_data_query_status2(), but without the is_loading parameter. See Data Prefetch for more details.

57.6.5.47 starpu_data_set_reduction_methods()

```
void starpu_data_set_reduction_methods (
            starpu_data_handle_t handle,
            struct starpu_codelet * redux_cl,
            struct starpu_codelet * init_cl )
```

Set the codelets to be used for `handle` when it is accessed in the mode STARPU_REDUX. Per-worker buffers will be initialized with the codelet `init_cl` (which has to take one handle with STARPU_W), and reduction between per-worker buffers will be done with the codelet `redux_cl` (which has to take a first accumulation handle with STARPU_RW|STARPU_COMMUTE, and a second contribution handle with STARPU_R). See Data Reduction and Temporary Data for more details.

57.6.5.48 starpu_data_set_reduction_methods_with_args()

```
void starpu_data_set_reduction_methods_with_args (
            starpu_data_handle_t handle,
            struct starpu_codelet * redux_cl,
            void * redux_cl_arg,
            struct starpu_codelet * init_cl,
            void * init_cl_arg )
```

Same as starpu_data_set_reduction_methods() but allows to pass arguments to the reduction and init tasks

57.6.5.49 starpu_data_test_if_allocated_on_node()

```
unsigned starpu_data_test_if_allocated_on_node (
            starpu_data_handle_t handle,
            unsigned memory_node )
```

See Data Prefetch for more details.

57.6.5.50 starpu_data_test_if_mapped_on_node()

```
unsigned starpu_data_test_if_mapped_on_node (
            starpu_data_handle_t handle,
            unsigned memory_node )
```

See Data Prefetch for more details.

57.6.5.51 starpu_memchunk_tidy()

```
void starpu_memchunk_tidy (
            unsigned memory_node )
```

See Data Prefetch for more details.

57.6.5.52 starpu_data_set_user_data()

```
void starpu_data_set_user_data (
            starpu_data_handle_t handle,
            void * user_data )
```

Set the field `user_data` for the `handle` to `user_data` . It can then be retrieved with starpu_data_get_user_data(). `user_data` can be any application-defined value, for instance a pointer to an object-oriented container for the data. See Data handles helpers for more details.

57.6.5.53 starpu_data_get_user_data()

```
void * starpu_data_get_user_data (
            starpu_data_handle_t handle )
```

Retrieve the field `user_data` previously set for the `handle`. See Data handles helpers for more details.

57.6.5.54 starpu_data_set_sched_data()

```
void starpu_data_set_sched_data (
            starpu_data_handle_t handle,
            void * sched_data )
```

Set the field `sched_data` for the `handle` to `sched_data` . It can then be retrieved with starpu_data_get_sched_data(). `sched_data` can be any scheduler-defined value. See Data handles helpers for more details.

57.6.5.55 starpu_data_get_sched_data()

```
void * starpu_data_get_sched_data (
            starpu_data_handle_t handle )
```

Retrieve the field `sched_data` previously set for the `handle`. See Data handles helpers for more details.

57.6.5.56 starpu_data_can_evict()

```
int starpu_data_can_evict (
            starpu_data_handle_t handle,
            unsigned node,
            enum starpu_is_prefetch is_prefetch )
```

Check whether data `handle` can be evicted now from node `node`. See Data Prefetch for more details.

57.7 Data Partition

Data Structures

- struct starpu_data_filter

Basic API

- void starpu_data_partition (starpu_data_handle_t initial_handle, struct starpu_data_filter ∗f)
- void starpu_data_unpartition (starpu_data_handle_t root_data, unsigned gathering_node)
- starpu_data_handle_t starpu_data_get_child (starpu_data_handle_t handle, unsigned i)
- int starpu_data_get_nb_children (starpu_data_handle_t handle)
- starpu_data_handle_t starpu_data_get_sub_data (starpu_data_handle_t root_data, unsigned depth,...)
- starpu_data_handle_t starpu_data_vget_sub_data (starpu_data_handle_t root_data, unsigned depth, va_list pa)
- void starpu_data_map_filters (starpu_data_handle_t root_data, unsigned nfilters,...)
- void starpu_data_vmap_filters (starpu_data_handle_t root_data, unsigned nfilters, va_list pa)
- void starpu_data_map_filters_parray (starpu_data_handle_t root_handle, int nfilters, struct starpu_data_filter ∗∗filters)
- void starpu_data_map_filters_array (starpu_data_handle_t root_handle, int nfilters, struct starpu_data_filter ∗filters)

Asynchronous API

- void starpu_data_partition_plan (starpu_data_handle_t initial_handle, struct starpu_data_filter ∗f, starpu_data_handle_t ∗children)
- void starpu_data_partition_submit (starpu_data_handle_t initial_handle, unsigned nparts, starpu_data_handle_t ∗children)
- void starpu_data_partition_readonly_submit (starpu_data_handle_t initial_handle, unsigned nparts, starpu_data_handle_t ∗children)
- void starpu_data_partition_readonly_submit_sequential_consistency (starpu_data_handle_t initial_handle, unsigned nparts, starpu_data_handle_t ∗children, int sequential_consistency)
- void starpu_data_partition_readwrite_upgrade_submit (starpu_data_handle_t initial_handle, unsigned nparts, starpu_data_handle_t ∗children)
- void starpu_data_partition_readonly_downgrade_submit (starpu_data_handle_t initial_handle, unsigned nparts, starpu_data_handle_t ∗children)
- void starpu_data_unpartition_submit (starpu_data_handle_t initial_handle, unsigned nparts, starpu_data_handle_t ∗children, int gathering_node)
- void starpu_data_unpartition_readonly_submit (starpu_data_handle_t initial_handle, unsigned nparts, starpu_data_handle_t ∗children, int gathering_node)
- void starpu_data_partition_clean (starpu_data_handle_t root_data, unsigned nparts, starpu_data_handle_t ∗children)
- void starpu_data_partition_clean_node (starpu_data_handle_t root_data, unsigned nparts, starpu_data_handle_t ∗children, int gather_node)
- void starpu_data_unpartition_submit_sequential_consistency_cb (starpu_data_handle_t initial_handle, unsigned nparts, starpu_data_handle_t ∗children, int gather_node, int sequential_consistency, void(∗callback↩_func)(void ∗), void ∗callback_arg)
- void starpu_data_partition_submit_sequential_consistency (starpu_data_handle_t initial_handle, unsigned nparts, starpu_data_handle_t ∗children, int sequential_consistency)
- void starpu_data_unpartition_submit_sequential_consistency (starpu_data_handle_t initial_handle, unsigned nparts, starpu_data_handle_t ∗children, int gathering_node, int sequential_consistency)

Predefined BCSR Filter Functions

Predefined partitioning functions for BCSR data. Examples on how to use them are shown in Partitioning Data.

- void starpu_bcsr_filter_canonical_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)

- unsigned starpu_bcsr_filter_canonical_block_get_nchildren (struct starpu_data_filter ∗f, starpu_data_handle_t handle)
- struct starpu_data_interface_ops ∗ starpu_bcsr_filter_canonical_block_child_ops (struct starpu_data_filter ∗f, unsigned child)
- void starpu_bcsr_filter_vertical_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)

Predefined CSR Filter Functions

Predefined partitioning functions for CSR data. Examples on how to use them are shown in Partitioning Data.

- void starpu_csr_filter_vertical_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)

Predefined Matrix Filter Functions

Predefined partitioning functions for matrix data. Examples on how to use them are shown in Partitioning Data. Note: this is using the C element order which is row-major, i.e. elements with consecutive x coordinates are consecutive in memory.

- void starpu_matrix_filter_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_matrix_filter_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_matrix_filter_vertical_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_matrix_filter_vertical_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_matrix_filter_pick_vector_y (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- struct starpu_data_interface_ops ∗ starpu_matrix_filter_pick_vector_child_ops (struct starpu_data_filter ∗f, unsigned child)
- void starpu_matrix_filter_pick_variable (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- struct starpu_data_interface_ops ∗ starpu_matrix_filter_pick_variable_child_ops (struct starpu_data_filter ∗f, unsigned child)

Predefined Vector Filter Functions

Predefined partitioning functions for vector data. Examples on how to use them are shown in Partitioning Data.

- void starpu_vector_filter_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_vector_filter_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_vector_filter_list_long (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_vector_filter_list (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_vector_filter_divide_in_2 (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_vector_filter_pick_variable (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- struct starpu_data_interface_ops ∗ starpu_vector_filter_pick_variable_child_ops (struct starpu_data_filter ∗f, unsigned child)

Predefined Block Filter Functions

Predefined partitioning functions for block data. Examples on how to use them are shown in Partitioning Data. An example is available in `examples/filters/shadow3d.c` Note: this is using the C element order which is row-major, i.e. elements with consecutive x coordinates are consecutive in memory.

- void starpu_block_filter_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_block_filter_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_block_filter_vertical_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_block_filter_vertical_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_block_filter_depth_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_block_filter_depth_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_block_filter_pick_matrix_z (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_block_filter_pick_matrix_y (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- struct starpu_data_interface_ops ∗ starpu_block_filter_pick_matrix_child_ops (struct starpu_data_filter ∗f, unsigned child)
- void starpu_block_filter_pick_variable (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- struct starpu_data_interface_ops ∗ starpu_block_filter_pick_variable_child_ops (struct starpu_data_filter ∗f, unsigned child)

Predefined Tensor Filter Functions

Predefined partitioning functions for tensor data.

- void starpu_tensor_filter_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_vertical_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_vertical_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_depth_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_depth_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_time_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_time_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_pick_block_t (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_pick_block_z (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_pick_block_y (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- struct starpu_data_interface_ops ∗ starpu_tensor_filter_pick_block_child_ops (struct starpu_data_filter ∗f, unsigned child)

- void starpu_tensor_filter_pick_variable (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- struct starpu_data_interface_ops ∗ starpu_tensor_filter_pick_variable_child_ops (struct starpu_data_filter ∗f, unsigned child)

Predefined Ndim Filter Functions

Predefined partitioning functions for ndim array data.

- void starpu_ndim_filter_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_to_tensor (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_to_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_to_matrix (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_to_vector (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_to_variable (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_pick_ndim (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_5d_pick_tensor (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_4d_pick_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_3d_pick_matrix (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_2d_pick_vector (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_1d_pick_variable (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_pick_variable (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_pick_tensor_child_ops (struct starpu_data_filter ∗f, unsigned child)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_pick_block_child_ops (struct starpu_data_filter ∗f, unsigned child)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_pick_matrix_child_ops (struct starpu_data_filter ∗f, unsigned child)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_pick_vector_child_ops (struct starpu_data_filter ∗f, unsigned child)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_pick_variable_child_ops (struct starpu_data_filter ∗f, unsigned child)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_to_tensor_child_ops (struct starpu_data_filter ∗f, unsigned child)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_to_block_child_ops (struct starpu_data_filter ∗f, unsigned child)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_to_matrix_child_ops (struct starpu_data_filter ∗f, unsigned child)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_to_vector_child_ops (struct starpu_data_filter ∗f, unsigned child)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_to_variable_child_ops (struct starpu_data_filter ∗f, unsigned child)
- void starpu_filter_nparts_compute_chunk_size_and_offset (unsigned n, unsigned nparts, size_t elemsize, unsigned id, unsigned blocksize, unsigned ∗chunk_size, size_t ∗offset)

57.7.1 Detailed Description

57.7.2 Data Structure Documentation

57.7.2.1 struct starpu_data_filter

Describe a data partitioning operation, to be given to starpu_data_partition(). See Defining A New Data Filter for more details.

Data Fields

- void(∗ filter_func)(void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗, unsigned id, unsigned nparts)
- unsigned nchildren
- unsigned(∗ get_nchildren)(struct starpu_data_filter ∗, starpu_data_handle_t initial_handle)
- struct starpu_data_interface_ops ∗(∗ get_child_ops)(struct starpu_data_filter ∗, unsigned id)
- unsigned filter_arg
- void ∗ filter_arg_ptr

57.7.2.1.1 Field Documentation

57.7.2.1.1.1 filter_func `void(* starpu_data_filter::filter_func) (void *father_interface, void *child_interface, struct starpu_data_filter *, unsigned id, unsigned nparts)`
Fill the `child_interface` structure with interface information for the `i` -th child of the parent `father_interface` (among `nparts`). The `filter` structure is provided, allowing to inspect the starpu_data_filter::filter_arg and starpu_data_filter::filter_arg_ptr parameters. The details of what needs to be filled in `child_interface` vary according to the data interface, but generally speaking:

- `id` is usually just copied over from the father, when the sub data has the same structure as the father, e.g. a subvector is a vector, a submatrix is a matrix, etc. This is however not the case for instance when dividing a BCSR matrix into its dense blocks, which then are matrices.

- `nx`, `ny` and alike are usually divided by the number of subdata, depending how the subdivision is done (e.g. nx division vs ny division for vertical matrix division vs horizontal matrix division).

- `ld` for matrix interfaces are usually just copied over: the leading dimension (ld) usually does not change.

- `elemsize` is usually just copied over.

- `ptr`, the pointer to the data, has to be computed according to `i` and the father's `ptr`, so as to point to the start of the sub data. This should however be done only if the father has `ptr` different from NULL: in the OpenCL case notably, the `dev_handle` and `offset` fields are used instead.

- `dev_handle` should be just copied over from the parent.

- `offset` has to be computed according to `i` and the father's `offset`, so as to provide the offset of the start of the sub data. This is notably used for the OpenCL case.

57.7.2.1.1.2 nchildren `unsigned starpu_data_filter::nchildren`
Number of parts to partition the data into.

57.7.2.1.1.3 get_nchildren `unsigned(* starpu_data_filter::get_nchildren) (struct starpu_data_filter *, starpu_data_handle_t initial_handle)`
Return the number of children. This can be used instead of starpu_data_filter::nchildren when the number of children depends on the actual data (e.g. the number of blocks in a sparse matrix).

57.7.2.1.1.4 get_child_ops `struct starpu_data_interface_ops *(* starpu_data_filter::get_child_ops) (struct starpu_data_filter *, unsigned id)`
When children use different data interface, return which interface is used by child number `id`.

57.7.2.1.1.5 filter_arg `unsigned starpu_data_filter::filter_arg`

Additional parameter for the filter function

57.7.2.1.1.6 filter_arg_ptr `void* starpu_data_filter::filter_arg_ptr`

Additional pointer parameter for the filter function, such as the sizes of the different parts.

57.7.3 Function Documentation

57.7.3.1 starpu_data_partition()

```
void starpu_data_partition (
             starpu_data_handle_t initial_handle,
             struct starpu_data_filter * f )
```

Request the partitioning of `initial_handle` into several subdata according to the filter `f`.

Here an example of how to use the function.

```
struct starpu_data_filter f =
{
  .filter_func = starpu_matrix_filter_block,
  .nchildren = nslicesx
};
starpu_data_partition(A_handle, &f);
```

See Partitioning Data for more details.

57.7.3.2 starpu_data_unpartition()

```
void starpu_data_unpartition (
             starpu_data_handle_t root_data,
             unsigned gathering_node )
```

Unapply the filter which has been applied to `root_data`, thus unpartitioning the data. The pieces of data are collected back into one big piece in the `gathering_node` (usually STARPU_MAIN_RAM). Tasks working on the partitioned data will be waited for by starpu_data_unpartition().

Here an example of how to use the function.

```
starpu_data_unpartition(A_handle, STARPU_MAIN_RAM);
```

See Partitioning Data for more details.

57.7.3.3 starpu_data_get_child()

```
starpu_data_handle_t starpu_data_get_child (
             starpu_data_handle_t handle,
             unsigned i )
```

Return the `i` -th child of the given `handle`, which must have been partitioned beforehand. See Partitioning Data for more details.

57.7.3.4 starpu_data_get_nb_children()

```
int starpu_data_get_nb_children (
             starpu_data_handle_t handle )
```

Return the number of children `handle` has been partitioned into. See Partitioning Data for more details.

57.7.3.5 starpu_data_get_sub_data()

```
starpu_data_handle_t starpu_data_get_sub_data (
             starpu_data_handle_t root_data,
             unsigned depth,
              ... )
```

After partitioning a StarPU data by applying a filter, starpu_data_get_sub_data() can be used to get handles for each of the data portions. `root_data` is the parent data that was partitioned. `depth` is the number of filters to traverse (in case several filters have been applied, to e.g. partition in row blocks, and then in column blocks), and the subsequent parameters are the indexes. The function returns a handle to the subdata.

Here an example of how to use the function.

```
h = starpu_data_get_sub_data(A_handle, 1, taskx);
```

See Partitioning Data for more details.

57.7.3.6 starpu_data_vget_sub_data()

```
starpu_data_handle_t starpu_data_vget_sub_data (
            starpu_data_handle_t root_data,
            unsigned depth,
            va_list pa )
```

Similar to starpu_data_get_sub_data() but use a `va_list` for the parameter list. See Partitioning Data for more details.

57.7.3.7 starpu_data_map_filters()

```
void starpu_data_map_filters (
            starpu_data_handle_t root_data,
            unsigned nfilters,
             ... )
```

Apply `nfilters` filters to the handle designated by `root_handle` recursively. `nfilters` pointers to variables of the type starpu_data_filter should be given. See Partitioning Data for more details.

57.7.3.8 starpu_data_vmap_filters()

```
void starpu_data_vmap_filters (
            starpu_data_handle_t root_data,
            unsigned nfilters,
            va_list pa )
```

Apply `nfilters` filters to the handle designated by `root_handle` recursively. Use a `va_list` of pointers to variables of the type starpu_data_filter. See Partitioning Data for more details.

57.7.3.9 starpu_data_map_filters_parray()

```
void starpu_data_map_filters_parray (
            starpu_data_handle_t root_handle,
            int nfilters,
            struct starpu_data_filter ** filters )
```

Apply `nfilters` filters to the handle designated by `root_handle` recursively. The pointer of the filter list `filters` of the type starpu_data_filter should be given. See Partitioning Data for more details.

57.7.3.10 starpu_data_map_filters_array()

```
void starpu_data_map_filters_array (
            starpu_data_handle_t root_handle,
            int nfilters,
            struct starpu_data_filter * filters )
```

Apply `nfilters` filters to the handle designated by `root_handle` recursively. The list of filter `filters` of the type starpu_data_filter should be given. See Partitioning Data for more details.

57.7.3.11 starpu_data_partition_plan()

```
void starpu_data_partition_plan (
            starpu_data_handle_t initial_handle,
            struct starpu_data_filter * f,
            starpu_data_handle_t * children )
```

Plan to partition `initial_handle` into several subdata according to the filter `f`. The handles are returned into the `children` array, which has to be the same size as the number of parts described in `f`. These handles are not immediately usable, starpu_data_partition_submit() has to be called to submit the actual partitioning.

Here is an example of how to use the function:

```
starpu_data_handle_t children[nslicesx];
```

```
struct starpu_data_filter f =
{
  .filter_func = starpu_matrix_filter_block,
  .nchildren = nslicesx
  };
  starpu_data_partition_plan(A_handle, &f, children);
```

See Asynchronous Partitioning for more details.

57.7.3.12 starpu_data_partition_submit()

```
void starpu_data_partition_submit (
            starpu_data_handle_t initial_handle,
            unsigned nparts,
            starpu_data_handle_t * children )
```

Submit the actual partitioning of `initial_handle` into the `nparts` `children` handles. This call is asynchronous, it only submits that the partitioning should be done, so that the `children` handles can now be used to submit tasks, and `initial_handle` can not be used to submit tasks any more (to guarantee coherency). For instance,

```
starpu_data_partition_submit(A_handle, nslicesx, children);
```

See Asynchronous Partitioning for more details.

57.7.3.13 starpu_data_partition_readonly_submit()

```
void starpu_data_partition_readonly_submit (
            starpu_data_handle_t initial_handle,
            unsigned nparts,
            starpu_data_handle_t * children )
```

Similar to starpu_data_partition_submit(), but do not invalidate `initial_handle`. This allows to continue using it, but the application has to be careful not to write to `initial_handle` or `children` handles, only read from them, since the coherency is otherwise not guaranteed. This thus allows to submit various tasks which concurrently read from various partitions of the data.

When the application wants to write to `initial_handle` again, it should call starpu_data_unpartition_submit(), which will properly add dependencies between the reads on the `children` and the writes to be submitted.

If instead the application wants to write to `children` handles, it should call starpu_data_partition_readwrite_upgrade_submit(), which will correctly add dependencies between the reads on the `initial_handle` and the writes to be submitted. See Asynchronous Partitioning for more details.

57.7.3.14 starpu_data_partition_readonly_submit_sequential_consistency()

```
void starpu_data_partition_readonly_submit_sequential_consistency (
            starpu_data_handle_t initial_handle,
            unsigned nparts,
            starpu_data_handle_t * children,
            int sequential_consistency )
```

Similar to starpu_data_partition_readonly_submit(), but allow to specify the coherency to be used for the main data `initial_handle`. See Asynchronous Partitioning for more details.

57.7.3.15 starpu_data_partition_readwrite_upgrade_submit()

```
void starpu_data_partition_readwrite_upgrade_submit (
            starpu_data_handle_t initial_handle,
            unsigned nparts,
            starpu_data_handle_t * children )
```

Assume that a partitioning of `initial_handle` has already been submitted in readonly mode through starpu_data_partition_readonly_submit(), and will upgrade that partitioning into read-write mode for the `children`, by invalidating `initial_handle`, and adding the necessary dependencies. See Asynchronous Partitioning for more details.

57.7.3.16 starpu_data_partition_readonly_downgrade_submit()

```
void starpu_data_partition_readonly_downgrade_submit (
            starpu_data_handle_t initial_handle,
            unsigned nparts,
            starpu_data_handle_t * children )
```

Assume that a partitioning of `initial_handle` has already been submitted in read-write mode through starpu_data_partition_submit(), and will downgrade that partitioning into read-only mode for the `children`, fetching data back to the `initial_handle`, and adding the necessary dependencies. See Asynchronous Partitioning for more details.

57.7.3.17 starpu_data_unpartition_submit()

```
void starpu_data_unpartition_submit (
            starpu_data_handle_t initial_handle,
            unsigned nparts,
            starpu_data_handle_t * children,
            int gathering_node )
```

Assuming that `initial_handle` is partitioned into `children`, submit an unpartitionning of `initial_handle`, i.e. submit a gathering of the pieces on the requested `gathering_node` memory node, and submit an invalidation of the children. See Asynchronous Partitioning for more details.

57.7.3.18 starpu_data_unpartition_readonly_submit()

```
void starpu_data_unpartition_readonly_submit (
            starpu_data_handle_t initial_handle,
            unsigned nparts,
            starpu_data_handle_t * children,
            int gathering_node )
```

Similar to starpu_data_partition_submit(), but do not invalidate `initial_handle`. This allows to continue using it, but the application has to be careful not to write to `initial_handle` or `children` handles, only read from them, since the coherency is otherwise not guaranteed. This thus allows to submit various tasks which concurrently read from various partitions of the data. See Asynchronous Partitioning for more details.

57.7.3.19 starpu_data_partition_clean()

```
void starpu_data_partition_clean (
            starpu_data_handle_t root_data,
            unsigned nparts,
            starpu_data_handle_t * children )
```

Clear the partition planning established between `root_data` and `children` with starpu_data_partition_plan(). This will notably submit an unregister all the `children`, which can thus not be used any more afterwards. See Asynchronous Partitioning for more details.

57.7.3.20 starpu_data_partition_clean_node()

```
void starpu_data_partition_clean_node (
            starpu_data_handle_t root_data,
            unsigned nparts,
            starpu_data_handle_t * children,
            int gather_node )
```

Similar to starpu_data_partition_clean() but the root data will be gathered on the given node. See Asynchronous Partitioning for more details.

57.7.3.21 starpu_data_unpartition_submit_sequential_consistency_cb()

```
void starpu_data_unpartition_submit_sequential_consistency_cb (
            starpu_data_handle_t initial_handle,
            unsigned nparts,
            starpu_data_handle_t * children,
            int gather_node,
            int sequential_consistency,
            void(*)(void *) callback_func,
            void * callback_arg )
```

Similar to starpu_data_unpartition_submit_sequential_consistency() but allow to specify a callback function for the unpartitiong task. See Asynchronous Partitioning for more details.

57.7.3.22 starpu_data_partition_submit_sequential_consistency()

```
void starpu_data_partition_submit_sequential_consistency (
             starpu_data_handle_t initial_handle,
             unsigned nparts,
             starpu_data_handle_t * children,
             int sequential_consistency )
```

Similar to starpu_data_partition_submit() but also allow to specify the coherency to be used for the main data `initial_handle` through the parameter `sequential_consistency`. See Asynchronous Partitioning for more details.

57.7.3.23 starpu_data_unpartition_submit_sequential_consistency()

```
void starpu_data_unpartition_submit_sequential_consistency (
             starpu_data_handle_t initial_handle,
             unsigned nparts,
             starpu_data_handle_t * children,
             int gathering_node,
             int sequential_consistency )
```

Similar to starpu_data_unpartition_submit() but also allow to specify the coherency to be used for the main data `initial_handle` through the parameter `sequential_consistency`. See Asynchronous Partitioning for more details.

57.7.3.24 starpu_bcsr_filter_canonical_block()

```
void starpu_bcsr_filter_canonical_block (
             void * father_interface,
             void * child_interface,
             struct starpu_data_filter * f,
             unsigned id,
             unsigned nparts )
```

Partition a block-sparse matrix into dense matrices. starpu_data_filter::get_child_ops needs to be set to starpu_bcsr_filter_canonical_block_child_ops() and starpu_data_filter::get_nchildren set to starpu_bcsr_filter_canonical_block_get_n
See BCSR Data Interface for more details.

57.7.3.25 starpu_bcsr_filter_canonical_block_get_nchildren()

```
unsigned starpu_bcsr_filter_canonical_block_get_nchildren (
             struct starpu_data_filter * f,
             starpu_data_handle_t handle )
```

Return the number of children obtained with starpu_bcsr_filter_canonical_block(). See BCSR Data Interface for more details.

57.7.3.26 starpu_bcsr_filter_canonical_block_child_ops()

```
struct starpu_data_interface_ops * starpu_bcsr_filter_canonical_block_child_ops (
             struct starpu_data_filter * f,
             unsigned child )
```

Return the child_ops of the partition obtained with starpu_bcsr_filter_canonical_block(). See BCSR Data Interface for more details.

57.7.3.27 starpu_bcsr_filter_vertical_block()

```
void starpu_bcsr_filter_vertical_block (
             void * father_interface,
             void * child_interface,
             struct starpu_data_filter * f,
             unsigned id,
             unsigned nparts )
```

Partition a block-sparse matrix into block-sparse matrices.
The split is done along the leading dimension, i.e. along adjacent nnz blocks.
See BCSR Data Interface for more details.

57.7.3.28 starpu_csr_filter_vertical_block()

```
void starpu_csr_filter_vertical_block (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a block-sparse matrix into vertical block-sparse matrices.
See CSR Data Interface for more details.

57.7.3.29 starpu_matrix_filter_block()

```
void starpu_matrix_filter_block (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a dense Matrix along the x dimension, thus getting (x/`nparts` ,y) matrices. If `nparts` does not divide x, the last submatrix contains the remainder.
See Matrix Data Interface for more details.

57.7.3.30 starpu_matrix_filter_block_shadow()

```
void starpu_matrix_filter_block_shadow (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a dense Matrix along the x dimension, with a shadow border starpu_data_filter::filter_arg_ptr, thus getting ((x-2∗shadow)/`nparts` +2∗shadow,y) matrices. If `nparts` does not divide x-2∗shadow, the last submatrix contains the remainder.
IMPORTANT: This can only be used for read-only access, as no coherency is enforced for the shadowed parts. A usage example is available in examples/filters/shadow2d.c
See Matrix Data Interface for more details.

57.7.3.31 starpu_matrix_filter_vertical_block()

```
void starpu_matrix_filter_vertical_block (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a dense Matrix along the y dimension, thus getting (x,y/`nparts`) matrices. If `nparts` does not divide y, the last submatrix contains the remainder.
See Matrix Data Interface for more details.

57.7.3.32 starpu_matrix_filter_vertical_block_shadow()

```
void starpu_matrix_filter_vertical_block_shadow (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a dense Matrix along the y dimension, with a shadow border starpu_data_filter::filter_arg_ptr, thus getting (x,(y-2∗shadow)/`nparts` +2∗shadow) matrices. If `nparts` does not divide y-2∗shadow, the last submatrix contains the remainder.

IMPORTANT: This can only be used for read-only access, as no coherency is enforced for the shadowed parts. A usage example is available in examples/filters/shadow2d.c
See Matrix Data Interface for more details.

57.7.3.33 starpu_matrix_filter_pick_vector_y()

```
void starpu_matrix_filter_pick_vector_y (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Pick `nparts` contiguous vectors from a matrix along the Y dimension. The starting position on Y-axis is set in starpu_data_filter::filter_arg_ptr.
starpu_data_filter::get_child_ops needs to be set to starpu_matrix_filter_pick_vector_child_ops(). A usage example is available in examples/filters/fmatrix_pick_vector.c
See Matrix Data Interface for more details.

57.7.3.34 starpu_matrix_filter_pick_vector_child_ops()

```
struct starpu_data_interface_ops * starpu_matrix_filter_pick_vector_child_ops (
            struct starpu_data_filter * f,
            unsigned child )
```

Return the child_ops of the partition obtained with starpu_matrix_filter_pick_vector_y(). See Matrix Data Interface for more details.

57.7.3.35 starpu_matrix_filter_pick_variable()

```
void starpu_matrix_filter_pick_variable (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Pick `nparts` contiguous variables from a matrix. The starting position is set in starpu_data_filter::filter_arg_ptr.
starpu_data_filter::get_child_ops needs to be set to starpu_matrix_filter_pick_variable_child_ops(). A usage example is available in examples/filters/fmatrix_pick_variable.c
See Matrix Data Interface for more details.

57.7.3.36 starpu_matrix_filter_pick_variable_child_ops()

```
struct starpu_data_interface_ops * starpu_matrix_filter_pick_variable_child_ops (
            struct starpu_data_filter * f,
            unsigned child )
```

Return the child_ops of the partition obtained with starpu_matrix_filter_pick_variable(). See Matrix Data Interface for more details.

57.7.3.37 starpu_vector_filter_block()

```
void starpu_vector_filter_block (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Return in `child_interface` the `id` th element of the vector represented by `father_interface` once partitioned in `nparts` chunks of equal size.
See Vector Data Interface for more details.

57.7.3.38 starpu_vector_filter_block_shadow()

```
void starpu_vector_filter_block_shadow (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Return in `child_interface` the `id` th element of the vector represented by `father_interface` once partitioned in `nparts` chunks of equal size with a shadow border starpu_data_filter::filter_arg_ptr, thus getting a vector of size `(n-2*shadow)/nparts+2*shadow`. The starpu_data_filter::filter_arg_ptr field of `f` must be the shadow size casted into `void*`.
IMPORTANT: This can only be used for read-only access, as no coherency is enforced for the shadowed parts. An usage example is available in examples/filters/shadow.c
See Vector Data Interface for more details.

57.7.3.39 starpu_vector_filter_list_long()

```
void starpu_vector_filter_list_long (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Return in `child_interface` the `id` th element of the vector represented by `father_interface` once partitioned into `nparts` chunks according to the starpu_data_filter::filter_arg_ptr field of `f`. The starpu_data_filter::filter_arg_ptr field must point to an array of `nparts` long elements, each of which specifies the number of elements in each chunk of the partition.
See Vector Data Interface for more details.

57.7.3.40 starpu_vector_filter_list()

```
void starpu_vector_filter_list (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Return in `child_interface` the `id` th element of the vector represented by `father_interface` once partitioned into `nparts` chunks according to the starpu_data_filter::filter_arg_ptr field of `f`. The starpu_data_filter::filter_arg_ptr field must point to an array of `nparts` uint32_t elements, each of which specifies the number of elements in each chunk of the partition.
See Vector Data Interface for more details.

57.7.3.41 starpu_vector_filter_divide_in_2()

```
void starpu_vector_filter_divide_in_2 (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Return in `child_interface` the `id` th element of the vector represented by `father_interface` once partitioned in `2` chunks of equal size, ignoring nparts. Thus, `id` must be `0` or `1`.
See Vector Data Interface for more details.

57.7.3.42 starpu_vector_filter_pick_variable()

```
void starpu_vector_filter_pick_variable (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Pick `nparts` contiguous variables from a vector. The starting position is set in starpu_data_filter::filter_arg_ptr. starpu_data_filter::get_child_ops needs to be set to starpu_vector_filter_pick_variable_child_ops(). A usage example is available in examples/filters/fvector_pick_variable.c
See Vector Data Interface for more details.

57.7.3.43 starpu_vector_filter_pick_variable_child_ops()

```
struct starpu_data_interface_ops * starpu_vector_filter_pick_variable_child_ops (
            struct starpu_data_filter * f,
            unsigned child )
```

Return the child_ops of the partition obtained with starpu_vector_filter_pick_variable(). See Vector Data Interface for more details.

57.7.3.44 starpu_block_filter_block()

```
void starpu_block_filter_block (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a block along the X dimension, thus getting (x/`nparts` ,y,z) 3D matrices. If `nparts` does not divide x, the last submatrix contains the remainder.
See Block Data Interface for more details.

57.7.3.45 starpu_block_filter_block_shadow()

```
void starpu_block_filter_block_shadow (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a block along the X dimension, with a shadow border starpu_data_filter::filter_arg_ptr, thus getting ((x-2∗shadow)/`nparts` +2∗shadow,y,z) blocks. If `nparts` does not divide x, the last submatrix contains the remainder.
IMPORTANT: This can only be used for read-only access, as no coherency is enforced for the shadowed parts.
See Block Data Interface for more details.

57.7.3.46 starpu_block_filter_vertical_block()

```
void starpu_block_filter_vertical_block (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a block along the Y dimension, thus getting (x,y/`nparts` ,z) blocks. If `nparts` does not divide y, the last submatrix contains the remainder.
See Block Data Interface for more details.

57.7.3.47 starpu_block_filter_vertical_block_shadow()

```
void starpu_block_filter_vertical_block_shadow (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a block along the Y dimension, with a shadow border starpu_data_filter::filter_arg_ptr, thus getting (x,(y-2∗shadow)/`nparts` +2∗shadow,z) 3D matrices. If `nparts` does not divide y, the last submatrix contains the remainder.
IMPORTANT: This can only be used for read-only access, as no coherency is enforced for the shadowed parts.
See Block Data Interface for more details.

57.7.3.48 starpu_block_filter_depth_block()

```
void starpu_block_filter_depth_block (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a block along the Z dimension, thus getting (x,y,z/`nparts`) blocks. If `nparts` does not divide z, the last submatrix contains the remainder.
See Block Data Interface for more details.

57.7.3.49 starpu_block_filter_depth_block_shadow()

```
void starpu_block_filter_depth_block_shadow (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a block along the Z dimension, with a shadow border starpu_data_filter::filter_arg_ptr, thus getting (x,y,(z-2∗shadow)/`nparts` +2∗shadow) blocks. If `nparts` does not divide z, the last submatrix contains the remainder.
IMPORTANT: This can only be used for read-only access, as no coherency is enforced for the shadowed parts.
See Block Data Interface for more details.

57.7.3.50 starpu_block_filter_pick_matrix_z()

```
void starpu_block_filter_pick_matrix_z (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Pick `nparts` contiguous matrices from a block along the Z dimension. The starting position on Z-axis is set in starpu_data_filter::filter_arg_ptr.
starpu_data_filter::get_child_ops needs to be set to starpu_block_filter_pick_matrix_child_ops(). A usage example is available in examples/filters/fblock_pick_matrix.c

See Block Data Interface for more details.

57.7.3.51 starpu_block_filter_pick_matrix_y()

```
void starpu_block_filter_pick_matrix_y (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Pick `nparts` contiguous matrices from a block along the Y dimension. The starting position on Y-axis is set in starpu_data_filter::filter_arg_ptr.
starpu_data_filter::get_child_ops needs to be set to starpu_block_filter_pick_matrix_child_ops(). A usage example is available in examples/filters/fblock_pick_matrix.c
See Block Data Interface for more details.

57.7.3.52 starpu_block_filter_pick_matrix_child_ops()

```
struct starpu_data_interface_ops * starpu_block_filter_pick_matrix_child_ops (
            struct starpu_data_filter * f,
            unsigned child )
```

Return the child_ops of the partition obtained with starpu_block_filter_pick_matrix_z() and starpu_block_filter_pick_matrix_y().
See Block Data Interface for more details.

57.7.3.53 starpu_block_filter_pick_variable()

```
void starpu_block_filter_pick_variable (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Pick `nparts` contiguous variables from a block. The starting position is set in starpu_data_filter::filter_arg_ptr.
starpu_data_filter::get_child_ops needs to be set to starpu_block_filter_pick_variable_child_ops(). A usage example is available in examples/filters/fblock_pick_variable.c
See Block Data Interface for more details.

57.7.3.54 starpu_block_filter_pick_variable_child_ops()

```
struct starpu_data_interface_ops * starpu_block_filter_pick_variable_child_ops (
            struct starpu_data_filter * f,
            unsigned child )
```

Return the child_ops of the partition obtained with starpu_block_filter_pick_variable(). See Block Data Interface for more details.

57.7.3.55 starpu_tensor_filter_block()

```
void starpu_tensor_filter_block (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a tensor along the X dimension, thus getting (x/`nparts` ,y,z,t) tensors. If `nparts` does not divide x, the last submatrix contains the remainder.
See Tensor Data Interface for more details.

57.7.3.56 starpu_tensor_filter_block_shadow()

```
void starpu_tensor_filter_block_shadow (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a tensor along the X dimension, with a shadow border starpu_data_filter::filter_arg_ptr, thus getting ((x-2∗shadow)/`nparts` +2∗shadow,y,z,t) tensors. If `nparts` does not divide x, the last submatrix contains the remainder.

IMPORTANT: This can only be used for read-only access, as no coherency is enforced for the shadowed parts.
See Tensor Data Interface for more details.

57.7.3.57 starpu_tensor_filter_vertical_block()

```
void starpu_tensor_filter_vertical_block (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a tensor along the Y dimension, thus getting (x,y/`nparts` ,z,t) tensors. If `nparts` does not divide y, the last submatrix contains the remainder.
See Tensor Data Interface for more details.

57.7.3.58 starpu_tensor_filter_vertical_block_shadow()

```
void starpu_tensor_filter_vertical_block_shadow (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a tensor along the Y dimension, with a shadow border starpu_data_filter::filter_arg_ptr, thus getting (x,(y-2∗shadow)/`nparts` +2∗shadow,z,t) tensors. If `nparts` does not divide y, the last submatrix contains the remainder.

IMPORTANT: This can only be used for read-only access, as no coherency is enforced for the shadowed parts.
See Tensor Data Interface for more details.

57.7.3.59 starpu_tensor_filter_depth_block()

```
void starpu_tensor_filter_depth_block (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a tensor along the Z dimension, thus getting (x,y,z/`nparts`,t) tensors. If `nparts` does not divide z, the last submatrix contains the remainder.
See Tensor Data Interface for more details.

57.7.3.60 starpu_tensor_filter_depth_block_shadow()

```
void starpu_tensor_filter_depth_block_shadow (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a tensor along the Z dimension, with a shadow border starpu_data_filter::filter_arg_ptr, thus getting (x,y,(z-2∗shadow)/`nparts` +2∗shadow,t) tensors. If `nparts` does not divide z, the last submatrix contains the remainder.
IMPORTANT: This can only be used for read-only access, as no coherency is enforced for the shadowed parts.
See Tensor Data Interface for more details.

57.7.3.61 starpu_tensor_filter_time_block()

```
void starpu_tensor_filter_time_block (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a tensor along the T dimension, thus getting (x,y,z,t/`nparts`) tensors. If `nparts` does not divide t, the last submatrix contains the remainder.
See Tensor Data Interface for more details.

57.7.3.62 starpu_tensor_filter_time_block_shadow()

```
void starpu_tensor_filter_time_block_shadow (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a tensor along the T dimension, with a shadow border starpu_data_filter::filter_arg_ptr, thus getting (x,y,z,(t-2∗shadow)/`nparts` +2∗shadow) tensors. If `nparts` does not divide t, the last submatrix contains the remainder.
IMPORTANT: This can only be used for read-only access, as no coherency is enforced for the shadowed parts.
See Tensor Data Interface for more details.

57.7.3.63 starpu_tensor_filter_pick_block_t()

```
void starpu_tensor_filter_pick_block_t (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Pick `nparts` contiguous blocks from a tensor along the T dimension. The starting position on T-axis is set in starpu_data_filter::filter_arg_ptr.
starpu_data_filter::get_child_ops needs to be set to starpu_tensor_filter_pick_block_child_ops(). A usage example is available in examples/filters/ftensor_pick_block.c
See Tensor Data Interface for more details.

57.7.3.64 starpu_tensor_filter_pick_block_z()

```
void starpu_tensor_filter_pick_block_z (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Pick `nparts` contiguous blocks from a tensor along the Z dimension. The starting position on Z-axis is set in starpu_data_filter::filter_arg_ptr.
starpu_data_filter::get_child_ops needs to be set to starpu_tensor_filter_pick_block_child_ops(). A usage example is available in examples/filters/ftensor_pick_block.c
See Tensor Data Interface for more details.

57.7.3.65 starpu_tensor_filter_pick_block_y()

```
void starpu_tensor_filter_pick_block_y (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Pick `nparts` contiguous blocks from a tensor along the Y dimension. The starting position on Y-axis is set in starpu_data_filter::filter_arg_ptr.

starpu_data_filter::get_child_ops needs to be set to starpu_tensor_filter_pick_block_child_ops(). A usage example is available in examples/filters/ftensor_pick_block.c

See Tensor Data Interface for more details.

57.7.3.66 starpu_tensor_filter_pick_block_child_ops()

```
struct starpu_data_interface_ops * starpu_tensor_filter_pick_block_child_ops (
            struct starpu_data_filter * f,
            unsigned child )
```

Return the child_ops of the partition obtained with starpu_tensor_filter_pick_block_t(), starpu_tensor_filter_pick_block_z() and starpu_tensor_filter_pick_block_y(). See Tensor Data Interface for more details.

57.7.3.67 starpu_tensor_filter_pick_variable()

```
void starpu_tensor_filter_pick_variable (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Pick `nparts` contiguous variables from a tensor. The starting position is set in starpu_data_filter::filter_arg_ptr.

starpu_data_filter::get_child_ops needs to be set to starpu_tensor_filter_pick_variable_child_ops(). A usage example is available in examples/filters/ftensor_pick_variable.c

See Tensor Data Interface for more details.

57.7.3.68 starpu_tensor_filter_pick_variable_child_ops()

```
struct starpu_data_interface_ops * starpu_tensor_filter_pick_variable_child_ops (
            struct starpu_data_filter * f,
            unsigned child )
```

Return the child_ops of the partition obtained with starpu_tensor_filter_pick_variable(). See Tensor Data Interface for more details.

57.7.3.69 starpu_ndim_filter_block()

```
void starpu_ndim_filter_block (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a ndim array along the given dimension set in starpu_data_filter::filter_arg. If `nparts` does not divide the element number on dimension, the last submatrix contains the remainder.

See Ndim Data Interface for more details.

57.7.3.70 starpu_ndim_filter_block_shadow()

```
void starpu_ndim_filter_block_shadow (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a ndim array along the given dimension set in starpu_data_filter::filter_arg, with a shadow border starpu_data_filter::filter_arg_ptr. If `nparts` does not divide the element number on dimension, the last submatrix contains the remainder.

IMPORTANT: This can only be used for read-only access, as no coherency is enforced for the shadowed parts.

See Ndim Data Interface for more details.

57.7.3.71 starpu_ndim_filter_to_tensor()

```
void starpu_ndim_filter_to_tensor (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a 4-dim array into `nparts` tensors along the given dimension set in starpu_data_filter::filter_arg. starpu_data_filter::get_child_ops needs to be set to starpu_ndim_filter_to_tensor_child_ops(). A usage example is available in examples/filters/fndim_to_tensor.c

See Ndim Data Interface for more details.

57.7.3.72 starpu_ndim_filter_to_block()

```
void starpu_ndim_filter_to_block (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a 3-dim array into `nparts` blocks along the given dimension set in starpu_data_filter::filter_arg. starpu_data_filter::get_child_ops needs to be set to starpu_ndim_filter_to_block_child_ops(). A usage example is available in examples/filters/fndim_to_block.c

See Ndim Data Interface for more details.

57.7.3.73 starpu_ndim_filter_to_matrix()

```
void starpu_ndim_filter_to_matrix (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a 2-dim array into `nparts` matrices along the given dimension set in starpu_data_filter::filter_arg. starpu_data_filter::get_child_ops needs to be set to starpu_ndim_filter_to_matrix_child_ops(). A usage example is available in examples/filters/fndim_to_matrix.c

See Ndim Data Interface for more details.

57.7.3.74 starpu_ndim_filter_to_vector()

```
void starpu_ndim_filter_to_vector (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Partition a 1-dim array into `nparts` vectors.

starpu_data_filter::get_child_ops needs to be set to starpu_ndim_filter_to_vector_child_ops(). A usage example is available in examples/filters/fndim_to_vector.c
See Ndim Data Interface for more details.

57.7.3.75 starpu_ndim_filter_to_variable()

```
void starpu_ndim_filter_to_variable (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Transfer a 0-dim array to a variable.
starpu_data_filter::get_child_ops needs to be set to starpu_ndim_filter_to_variable_child_ops(). A usage example is available in examples/filters/fndim_to_variable.c
See Ndim Data Interface for more details.

57.7.3.76 starpu_ndim_filter_pick_ndim()

```
void starpu_ndim_filter_pick_ndim (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Pick `nparts` contiguous (n-1)dim arrays from a ndim array along the given dimension set in starpu_data_filter::filter_arg.
The starting position is set in starpu_data_filter::filter_arg_ptr.
A usage example is available in examples/filters/fndim_pick_ndim.c
See Ndim Data Interface for more details.

57.7.3.77 starpu_ndim_filter_5d_pick_tensor()

```
void starpu_ndim_filter_5d_pick_tensor (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Pick `nparts` contiguous tensors from a 5-dim array along the given dimension set in starpu_data_filter::filter_arg.
The starting position is set in starpu_data_filter::filter_arg_ptr.
starpu_data_filter::get_child_ops needs to be set to starpu_ndim_filter_pick_tensor_child_ops(). A usage example is available in examples/filters/fndim_5d_pick_tensor.c
See Ndim Data Interface for more details.

57.7.3.78 starpu_ndim_filter_4d_pick_block()

```
void starpu_ndim_filter_4d_pick_block (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Pick `nparts` contiguous blocks from a 4-dim array along the given dimension set in starpu_data_filter::filter_arg.
The starting position is set in starpu_data_filter::filter_arg_ptr.
starpu_data_filter::get_child_ops needs to be set to starpu_ndim_filter_pick_block_child_ops(). A usage example is available in examples/filters/fndim_4d_pick_block.c
See Ndim Data Interface for more details.

57.7.3.79 starpu_ndim_filter_3d_pick_matrix()

```
void starpu_ndim_filter_3d_pick_matrix (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Pick `nparts` contiguous matrices from a 3-dim array along the given dimension set in starpu_data_filter::filter_arg. The starting position is set in starpu_data_filter::filter_arg_ptr.
starpu_data_filter::get_child_ops needs to be set to starpu_ndim_filter_pick_matrix_child_ops(). A usage example is available in examples/filters/fndim_3d_pick_matrix.c
See Ndim Data Interface for more details.

57.7.3.80 starpu_ndim_filter_2d_pick_vector()

```
void starpu_ndim_filter_2d_pick_vector (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Pick `nparts` contiguous vectors from a 2-dim array along the given dimension set in starpu_data_filter::filter_arg. The starting position is set in starpu_data_filter::filter_arg_ptr.
starpu_data_filter::get_child_ops needs to be set to starpu_ndim_filter_pick_vector_child_ops(). A usage example is available in examples/filters/fndim_2d_pick_vector.c
See Ndim Data Interface for more details.

57.7.3.81 starpu_ndim_filter_1d_pick_variable()

```
void starpu_ndim_filter_1d_pick_variable (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Pick `nparts` contiguous variables from a 1-dim array. The starting position is set in starpu_data_filter::filter_arg_ptr.
starpu_data_filter::get_child_ops needs to be set to starpu_ndim_filter_pick_variable_child_ops(). A usage example is available in examples/filters/fndim_1d_pick_variable.c
See Ndim Data Interface for more details.

57.7.3.82 starpu_ndim_filter_pick_variable()

```
void starpu_ndim_filter_pick_variable (
            void * father_interface,
            void * child_interface,
            struct starpu_data_filter * f,
            unsigned id,
            unsigned nparts )
```

Pick `nparts` contiguous variables from a ndim array. The starting position is set in starpu_data_filter::filter_arg_ptr.
starpu_data_filter::get_child_ops needs to be set to starpu_ndim_filter_pick_variable_child_ops(). A usage example is available in examples/filters/fndim_pick_variable.c
See Ndim Data Interface for more details.

57.7.3.83 starpu_ndim_filter_pick_tensor_child_ops()

```
struct starpu_data_interface_ops * starpu_ndim_filter_pick_tensor_child_ops (
            struct starpu_data_filter * f,
            unsigned child )
```

Return the child_ops of the partition obtained with starpu_ndim_filter_pick_tensor(). See Ndim Data Interface for more details.

57.7.3.84 starpu_ndim_filter_pick_block_child_ops()

```
struct starpu_data_interface_ops * starpu_ndim_filter_pick_block_child_ops (
            struct starpu_data_filter * f,
            unsigned child )
```

Return the child_ops of the partition obtained with starpu_ndim_filter_pick_block(). See Ndim Data Interface for more details.

57.7.3.85 starpu_ndim_filter_pick_matrix_child_ops()

```
struct starpu_data_interface_ops * starpu_ndim_filter_pick_matrix_child_ops (
            struct starpu_data_filter * f,
            unsigned child )
```

Return the child_ops of the partition obtained with starpu_ndim_filter_pick_matrix(). See Ndim Data Interface for more details.

57.7.3.86 starpu_ndim_filter_pick_vector_child_ops()

```
struct starpu_data_interface_ops * starpu_ndim_filter_pick_vector_child_ops (
            struct starpu_data_filter * f,
            unsigned child )
```

Return the child_ops of the partition obtained with starpu_ndim_filter_pick_vector(). See Ndim Data Interface for more details.

57.7.3.87 starpu_ndim_filter_pick_variable_child_ops()

```
struct starpu_data_interface_ops * starpu_ndim_filter_pick_variable_child_ops (
            struct starpu_data_filter * f,
            unsigned child )
```

Return the child_ops of the partition obtained with starpu_ndim_filter_pick_variable(). See Ndim Data Interface for more details.

57.7.3.88 starpu_ndim_filter_to_tensor_child_ops()

```
struct starpu_data_interface_ops * starpu_ndim_filter_to_tensor_child_ops (
            struct starpu_data_filter * f,
            unsigned child )
```

Return the child_ops of the partition obtained with starpu_ndim_filter_to_tensor(). See Ndim Data Interface for more details.

57.7.3.89 starpu_ndim_filter_to_block_child_ops()

```
struct starpu_data_interface_ops * starpu_ndim_filter_to_block_child_ops (
            struct starpu_data_filter * f,
            unsigned child )
```

Return the child_ops of the partition obtained with starpu_ndim_filter_to_block(). See Ndim Data Interface for more details.

57.7.3.90 starpu_ndim_filter_to_matrix_child_ops()

```
struct starpu_data_interface_ops * starpu_ndim_filter_to_matrix_child_ops (
            struct starpu_data_filter * f,
            unsigned child )
```

Return the child_ops of the partition obtained with starpu_ndim_filter_to_matrix(). See Ndim Data Interface for more details.

57.7.3.91 starpu_ndim_filter_to_vector_child_ops()

```
struct starpu_data_interface_ops * starpu_ndim_filter_to_vector_child_ops (
            struct starpu_data_filter * f,
            unsigned child )
```

Return the child_ops of the partition obtained with starpu_ndim_filter_to_vector(). See Ndim Data Interface for more details.

57.7.3.92 starpu_ndim_filter_to_variable_child_ops()

```
struct starpu_data_interface_ops * starpu_ndim_filter_to_variable_child_ops (
            struct starpu_data_filter * f,
            unsigned child )
```

Return the child_ops of the partition obtained with starpu_ndim_filter_to_variable(). See Ndim Data Interface for more details.

57.7.3.93 starpu_filter_nparts_compute_chunk_size_and_offset()

```
void starpu_filter_nparts_compute_chunk_size_and_offset (
            unsigned n,
            unsigned nparts,
            size_t elemsize,
            unsigned id,
            unsigned blocksize,
            unsigned * chunk_size,
            size_t * offset )
```

Given an integer `n`, `n` the number of parts it must be divided in, `id` the part currently considered, determines the `chunk_size` and the `offset`, taking into account the size of the elements stored in the data structure `elemsize` and `blocksize`, which is most often 1. See Defining A New Data Filter for more details.

57.8 Expert Mode

Functions

- void starpu_wake_all_blocked_workers (void)
- int starpu_progression_hook_register (unsigned(∗func)(void ∗arg), void ∗arg)
- void starpu_progression_hook_deregister (int hook_id)
- int **starpu_idle_hook_register** (unsigned(∗func)(void ∗arg), void ∗arg)
- void **starpu_idle_hook_deregister** (int hook_id)

57.8.1 Detailed Description

57.8.2 Function Documentation

57.8.2.1 starpu_wake_all_blocked_workers()

```
void starpu_wake_all_blocked_workers (
            void  )
```

Wake all the workers, so they can inspect data requests and task submissions again.

57.8.2.2 starpu_progression_hook_register()

```
int starpu_progression_hook_register (
            unsigned(*)(void *arg) func,
            void * arg )
```

Register a progression hook, to be called when workers are idle.

57.8.2.3 starpu_progression_hook_deregister()

```
void starpu_progression_hook_deregister (
            int hook_id )
```

Unregister a given progression hook.

57.9 Explicit Dependencies

Typedefs

- typedef uint64_t starpu_tag_t

Functions

- void starpu_task_declare_deps_array (struct starpu_task ∗task, unsigned ndeps, struct starpu_task ∗task_array[ ])
- void starpu_task_declare_deps (struct starpu_task ∗task, unsigned ndeps,...)
- void starpu_task_declare_end_deps_array (struct starpu_task ∗task, unsigned ndeps, struct starpu_task ∗task_array[ ])
- void starpu_task_declare_end_deps (struct starpu_task ∗task, unsigned ndeps,...)
- int starpu_task_get_task_succs (struct starpu_task ∗task, unsigned ndeps, struct starpu_task ∗task_array[ ])
- int starpu_task_get_task_scheduled_succs (struct starpu_task ∗task, unsigned ndeps, struct starpu_task ∗task_array[ ])
- void starpu_task_end_dep_add (struct starpu_task ∗t, int nb_deps)
- void starpu_task_end_dep_release (struct starpu_task ∗t)
- void starpu_tag_declare_deps (starpu_tag_t id, unsigned ndeps,...)
- void starpu_tag_declare_deps_array (starpu_tag_t id, unsigned ndeps, starpu_tag_t ∗array)
- int starpu_tag_wait (starpu_tag_t id)
- int starpu_tag_wait_array (unsigned ntags, starpu_tag_t ∗id)
- void starpu_tag_restart (starpu_tag_t id)
- void starpu_tag_remove (starpu_tag_t id)
- void starpu_tag_notify_from_apps (starpu_tag_t id)
- void starpu_tag_notify_restart_from_apps (starpu_tag_t id)
- struct starpu_task ∗ starpu_tag_get_task (starpu_tag_t id)

57.9.1 Detailed Description

57.9.2 Typedef Documentation

57.9.2.1 starpu_tag_t

`typedef uint64_t starpu_tag_t`

Define a task logical identifier. It is possible to associate a task with a unique *tag* chosen by the application, and to express dependencies between tasks by the means of those tags. To do so, fill the field starpu_task::tag_id with a tag number (can be arbitrary) and set the field starpu_task::use_tag to 1. If starpu_tag_declare_deps() is called with this tag number, the task will not be started until the tasks which holds the declared dependency tags are completed.

57.9.3 Function Documentation

57.9.3.1 starpu_task_declare_deps_array()

```
void starpu_task_declare_deps_array (
            struct starpu_task * task,
            unsigned ndeps,
            struct starpu_task * task_array[ ] )
```

Declare task dependencies between a `task` and an array of tasks of length `ndeps`. This function must be called prior to the submission of the task, but it may called after the submission or the execution of the tasks in the array, provided the tasks are still valid (i.e. they were not automatically destroyed). Calling this function on a task that was already submitted or with an entry of `task_array` that is no longer a valid task results in an undefined behaviour.

If `ndeps` is 0, no dependency is added. It is possible to call starpu_task_declare_deps_array() several times on the same task, in this case, the dependencies are added. It is possible to have redundancy in the task dependencies. See Tasks And Tags Dependencies for more details.

57.9.3.2 starpu_task_declare_deps()

```
void starpu_task_declare_deps (
            struct starpu_task * task,
            unsigned ndeps,
             ... )
```

Declare task dependencies between a `task` and an series of `ndeps` tasks, similarly to starpu_task_declare_deps_array(), but the tasks are passed after `ndeps`, which indicates how many tasks `task` shall be made to depend on. If `ndeps` is 0, no dependency is added. See Tasks And Tags Dependencies for more details.

57.9.3.3 starpu_task_declare_end_deps_array()

```
void starpu_task_declare_end_deps_array (
            struct starpu_task * task,
            unsigned ndeps,
            struct starpu_task * task_array[] )
```

Declare task end dependencies between a `task` and an array of tasks of length `ndeps`. `task` will appear as terminated not only when `task` is termination, but also when the tasks of `task_array` have terminated. This function must be called prior to the termination of the task, but it may called after the submission or the execution of the tasks in the array, provided the tasks are still valid (i.e. they were not automatically destroyed). Calling this function on a task that was already terminated or with an entry of `task_array` that is no longer a valid task results in an undefined behaviour. If `ndeps` is 0, no dependency is added. It is possible to call starpu_task_declare_end_deps_array() several times on the same task, in this case, the dependencies are added. It is currently not implemented to have redundancy in the task dependencies. See Tasks And Tags Dependencies for more details.

57.9.3.4 starpu_task_declare_end_deps()

```
void starpu_task_declare_end_deps (
            struct starpu_task * task,
            unsigned ndeps,
             ... )
```

Declare task end dependencies between a `task` and an series of `ndeps` tasks, similarly to starpu_task_declare_end_deps_array(), but the tasks are passed after `ndeps`, which indicates how many tasks `task` 's termination shall be made to depend on. If `ndeps` is 0, no dependency is added. See Tasks And Tags Dependencies for more details.

57.9.3.5 starpu_task_get_task_succs()

```
int starpu_task_get_task_succs (
            struct starpu_task * task,
            unsigned ndeps,
            struct starpu_task * task_array[] )
```

Fill `task_array` with the list of tasks which are direct children of `task`. `ndeps` is the size of `task_array`. This function returns the number of direct children. `task_array` can be set to `NULL` if `ndeps` is 0, which allows to compute the number of children before allocating an array to store them. This function can only be called if `task` has not completed yet, otherwise the results are undefined. The result may also be outdated if some additional dependency has been added in the meanwhile. See Getting Task Children for more details.

57.9.3.6 starpu_task_get_task_scheduled_succs()

```
int starpu_task_get_task_scheduled_succs (
            struct starpu_task * task,
            unsigned ndeps,
            struct starpu_task * task_array[] )
```

Behave like starpu_task_get_task_succs(), except that it only reports tasks which will go through the scheduler, thus avoiding tasks with not codelet, or with explicit placement. See Getting Task Children for more details.

57.9.3.7 starpu_task_end_dep_add()

```
void starpu_task_end_dep_add (
            struct starpu_task * t,
            int nb_deps )
```

Add `nb_deps` end dependencies to the task `t`. This means the task will not terminate until the required number of calls to the function starpu_task_end_dep_release() has been made. See Tasks And Tags Dependencies for more details.

57.9.3.8 starpu_task_end_dep_release()

```
void starpu_task_end_dep_release (
            struct starpu_task * t )
```

Unlock 1 end dependency to the task `t`. This function must be called after starpu_task_end_dep_add(). See Tasks And Tags Dependencies for more details.

57.9.3.9 starpu_tag_declare_deps()

```
void starpu_tag_declare_deps (
            starpu_tag_t id,
            unsigned ndeps,
             ... )
```

Specify the dependencies of the task identified by tag `id`. The first argument specifies the tag which is configured, the second argument gives the number of tag(s) on which `id` depends. The following arguments are the tags which have to be terminated to unlock the task. This function must be called before the associated task is submitted to StarPU with starpu_task_submit().

WARNING! Use with caution. Because of the variable arity of starpu_tag_declare_deps(), note that the last arguments must be of type starpu_tag_t : constant values typically need to be explicitly casted. Otherwise, due to integer sizes and argument passing on the stack, the C compiler might consider the tag `0x200000003` instead of `0x2` and `0x3` when calling `starpu_tag_declare_deps(0x1, 2, 0x2, 0x3)`. Using the starpu_tag_declare_deps_array() function avoids this hazard.

```
//  Tag 0x1 depends on tags 0x32 and 0x52
starpu_tag_declare_deps((starpu_tag_t)0x1, 2, (starpu_tag_t)0x32, (starpu_tag_t)0x52);
```

See Tasks And Tags Dependencies for more details.

57.9.3.10 starpu_tag_declare_deps_array()

```
void starpu_tag_declare_deps_array (
            starpu_tag_t id,
            unsigned ndeps,
            starpu_tag_t * array )
```

Similar to starpu_tag_declare_deps(), except that its does not take a variable number of arguments but an `array` of tags of size `ndeps`.

```
// Tag 0x1 depends on tags 0x32 and 0x52
starpu_tag_t tag_array[2] = {0x32, 0x52};
starpu_tag_declare_deps_array((starpu_tag_t)0x1, 2, tag_array);
```

See Tasks And Tags Dependencies for more details.

57.9.3.11 starpu_tag_wait()

```
int starpu_tag_wait (
            starpu_tag_t id )
```

Block until the task associated to tag `id` has been executed. This is a blocking call which must therefore not be called within tasks or callbacks, but only from the application directly. It is possible to synchronize with the same tag multiple times, as long as the starpu_tag_remove() function is not called. Note that it is still possible to synchronize with a tag associated to a task for which the structure starpu_task was freed (e.g. if the field starpu_task::destroy was enabled). See Waiting For Tasks for more details.

57.9.3.12 starpu_tag_wait_array()

```
int starpu_tag_wait_array (
             unsigned ntags,
             starpu_tag_t * id )
```

Similar to starpu_tag_wait() except that it blocks until all the `ntags` tags contained in the array `id` are terminated. See Waiting For Tasks for more details.

57.9.3.13 starpu_tag_restart()

```
void starpu_tag_restart (
             starpu_tag_t id )
```

Clear the *already notified* status of a tag which is not associated with a task. Before that, calling starpu_tag_notify_from_apps() again will not notify the successors. After that, the next call to starpu_tag_notify_from_apps() will notify the successors. See Tasks And Tags Dependencies for more details.

57.9.3.14 starpu_tag_remove()

```
void starpu_tag_remove (
             starpu_tag_t id )
```

Release the resources associated to tag `id`. It can be called once the corresponding task has been executed and when there is no other tag that depend on this tag anymore. See Tasks And Tags Dependencies for more details.

57.9.3.15 starpu_tag_notify_from_apps()

```
void starpu_tag_notify_from_apps (
             starpu_tag_t id )
```

Explicitly unlock tag `id`. It may be useful in the case of applications which execute part of their computation outside StarPU tasks (e.g. third-party libraries). It is also provided as a convenient tool for the programmer, for instance to entirely construct the task DAG before actually giving StarPU the opportunity to execute the tasks. When called several times on the same tag, notification will be done only on first call, thus implementing "OR" dependencies, until the tag is restarted using starpu_tag_restart(). See Tasks And Tags Dependencies for more details.

57.9.3.16 starpu_tag_notify_restart_from_apps()

```
void starpu_tag_notify_restart_from_apps (
             starpu_tag_t id )
```

Atomically call starpu_tag_notify_from_apps() and starpu_tag_restart() on tag `id`. This is useful with cyclic graphs, when we want to safely trigger its startup. See Tasks And Tags Dependencies for more details.

57.9.3.17 starpu_tag_get_task()

```
struct starpu_task * starpu_tag_get_task (
             starpu_tag_t id )
```

Return the task associated to the tag `id`. See Tasks And Tags Dependencies for more details.

57.10 FFT Support

Functions

- void ∗ starpufft_malloc (size_t n)
- starpufft_plan starpufft_plan_dft_1d (int n, int sign, unsigned flags)
- starpufft_plan starpufft_plan_dft_2d (int n, int m, int sign, unsigned flags)
- struct starpu_task ∗ starpufft_start (starpufft_plan p, void ∗in, void ∗out)
- struct starpu_task ∗ starpufft_start_handle (starpufft_plan p, starpu_data_handle_t in, starpu_data_handle_t out)
- int starpufft_execute (starpufft_plan p, void ∗in, void ∗out)
- int starpufft_execute_handle (starpufft_plan p, starpu_data_handle_t in, starpu_data_handle_t out)
- void starpufft_cleanup (starpufft_plan p)
- void starpufft_destroy_plan (starpufft_plan p)

57.10.1 Detailed Description

57.10.2 Function Documentation

57.10.2.1 starpufft_malloc()

```
void * starpufft_malloc (
            size_t n )
```

Allocate memory for `n` bytes. This is preferred over `malloc()`, since it allocates pinned memory, which allows overlapped transfers.

57.10.2.2 starpufft_plan_dft_1d()

```
struct starpufft_plan * starpufft_plan_dft_1d (
            int n,
            int sign,
            unsigned flags )
```

Initialize a plan for 1D FFT of size `n`. `sign` can be STARPUFFT_FORWARD or STARPUFFT_INVERSE. `flags` must be 0.

57.10.2.3 starpufft_plan_dft_2d()

```
struct starpufft_plan * starpufft_plan_dft_2d (
            int n,
            int m,
            int sign,
            unsigned flags )
```

Initialize a plan for 2D FFT of size (`n`, `m`). `sign` can be STARPUFFT_FORWARD or STARPUFFT_INVERSE. flags must be `0`.

57.10.2.4 starpufft_start()

```
struct starpu_task * starpufft_start (
            starpufft_plan p,
            void * in,
            void * out )
```

Start an FFT previously planned as `p`, using `in` and `out` as input and output. This only submits the task and does not wait for it. The application should call starpufft_cleanup() to unregister the

57.10.2.5 starpufft_start_handle()

```
struct starpu_task * starpufft_start_handle (
            starpufft_plan p,
            starpu_data_handle_t in,
            starpu_data_handle_t out )
```

Start an FFT previously planned as `p`, using data handles `in` and `out` as input and output (assumed to be vectors of elements of the expected types). This only submits the task and does not wait for it.

57.10.2.6 starpufft_execute()

```
void starpufft_execute (
            starpufft_plan p,
            void * in,
            void * out )
```

Execute an FFT previously planned as `p`, using `in` and `out` as input and output. This submits and waits for the task.

57.10.2.7 starpufft_execute_handle()

```
void starpufft_execute_handle (
            starpufft_plan p,
            starpu_data_handle_t in,
            starpu_data_handle_t out )
```

Execute an FFT previously planned as `p`, using data handles `in` and `out` as input and output (assumed to be vectors of elements of the expected types). This submits and waits for the task.

57.10.2.8 starpufft_cleanup()

```
void starpufft_cleanup (
            starpufft_plan p )
```

Release data for plan `p`, in the starpufft_start() case.

57.10.2.9 starpufft_destroy_plan()

```
void starpufft_destroy_plan (
            starpufft_plan p )
```

Destroy plan `p`, i.e. release all CPU (fftw) and GPU (cufft) resources.

57.11 Fortran Support

Fortran API.

Namespaces

- module fstarpu_mod

  Fortran API.

57.11.1 Detailed Description

Fortran API.

57.12 FxT Support

Data Structures

- struct starpu_fxt_codelet_event
- struct starpu_fxt_mpi_offset
- struct starpu_fxt_options

Functions

- void **starpu_fxt_options_init** (struct starpu_fxt_options ∗options)
- void **starpu_fxt_options_shutdown** (struct starpu_fxt_options ∗options)
- void **starpu_fxt_generate_trace** (struct starpu_fxt_options ∗options)
- void starpu_fxt_autostart_profiling (int autostart)
- void starpu_fxt_start_profiling (void)
- void starpu_fxt_stop_profiling (void)
- void **starpu_fxt_write_data_trace** (char ∗filename_in)
- void **starpu_fxt_write_data_trace_in_dir** (char ∗filename_in, char ∗dir)
- int starpu_fxt_is_enabled (void)
- void starpu_fxt_trace_user_event (unsigned long code)
- void starpu_fxt_trace_user_event_string (const char ∗s)

57.12.1 Detailed Description

57.12.2 Data Structure Documentation

57.12.2.1 struct starpu_fxt_codelet_event

todo

Data Fields

char	symbol[2048]	
int	workerid	
char	perfmodel_archname[256]	
uint32_t	hash	
size_t	size	
float	time	

57.12.2.2 struct starpu_fxt_mpi_offset

Store information related to clock synchronizations: mainly the offset to apply to each time.

Data Fields

uint64_t	local_time_start	node time for the barrier at the beginning of the program
int64_t	offset_start	offset to apply to node time, computed at the beginning of the program
uint64_t	local_time_end	node time for the barrier at the end of the program (optional)
int64_t	offset_end	offset to apply to node time, computed at the end of the program (optional)
int	nb_barriers	number of barriers to synchronize clocks during the execution of the program (can be 0, 1 or 2)

57.12.2.3 struct starpu_fxt_options

todo

Data Fields

unsigned	per_task_colour	
unsigned	no_events	
unsigned	no_counter	
unsigned	no_bus	
unsigned	no_flops	
unsigned	ninputfiles	
unsigned	no_smooth	
unsigned	no_acquire	
unsigned	memory_states	
unsigned	internal	
unsigned	label_deps	
char ∗	filenames[STARPU_FXT_MAX_FILES]	
char ∗	out_paje_path	
char ∗	distrib_time_path	
char ∗	activity_path	
char ∗	sched_tasks_path	
char ∗	dag_path	
char ∗	tasks_path	
char ∗	data_path	
char ∗	papi_path	
char ∗	comms_path	
char ∗	number_events_path	
char ∗	anim_path	
char ∗	states_path	
char ∗	dir	
char	worker_names[STARPU_NMAXWORKERS][256]	
int	nworkers	
struct starpu_perfmodel_arch	worker_archtypes[STARPU_NMAXWORKERS]	
char ∗	file_prefix	In case we are going to gather multiple traces (e.g in the case of MPI processes), we may need to prefix the name of the containers.
struct starpu_fxt_mpi_offset	file_offset	In case we are going to gather multiple traces (e.g in the case of MPI processes), we may need to synchronize clocks and apply an offset.
int	file_rank	In case we are going to gather multiple traces (e.g in the case of MPI processes), this variable stores the MPI rank of the trace file.
struct starpu_fxt_codelet_event ∗∗	dumped_codelets	In case we want to dump the list of codelets to an external tool
long	dumped_codelets_count	In case we want to dump the list of codelets to an external tool, number of dumped codelets.

57.12.3 Function Documentation

57.12.3.1 starpu_fxt_autostart_profiling()

```
void starpu_fxt_autostart_profiling (
            int autostart )
```

Determine whether profiling should be started by starpu_init(), or only when starpu_fxt_start_profiling() is called. `autostart` should be 1 to do so, or 0 to prevent it. This function has to be called before starpu_init(). See Limiting The Scope Of The Trace for more details.

57.12.3.2 starpu_fxt_start_profiling()

```
void starpu_fxt_start_profiling (
            void )
```

Start recording the trace. The trace is by default started from starpu_init() call, but can be paused by using starpu_fxt_stop_profiling(), in which case starpu_fxt_start_profiling() should be called to resume recording events. See Limiting The Scope Of The Trace for more details.

57.12.3.3 starpu_fxt_stop_profiling()

```
void starpu_fxt_stop_profiling (
            void )
```

Stop recording the trace. The trace is by default stopped when calling starpu_shutdown(). starpu_fxt_stop_profiling() can however be used to stop it earlier. starpu_fxt_start_profiling() can then be called to start recording it again, etc. See Limiting The Scope Of The Trace for more details.

57.12.3.4 starpu_fxt_is_enabled()

```
int starpu_fxt_is_enabled (
            void )
```

Wrapper to get value of env variable STARPU_FXT_TRACE

57.12.3.5 starpu_fxt_trace_user_event()

```
void starpu_fxt_trace_user_event (
            unsigned long code )
```

Add an event in the execution trace if FxT is enabled. See Creating a Gantt Diagram for more details.

57.12.3.6 starpu_fxt_trace_user_event_string()

```
void starpu_fxt_trace_user_event_string (
            const char * s )
```

Add a string event in the execution trace if FxT is enabled. See Creating a Gantt Diagram for more details.

57.13 Heteroprio Scheduler

This is the interface for the heteroprio scheduler.

Macros

- #define **STARPU_HETEROPRIO_MAX_PREFETCH**
- #define **STARPU_AUTOHETEROPRIO_PRIORITY_ORDERING_POLICY_COUNT**

Enumerations

- enum starpu_autoheteroprio_priority_ordering_policy {
  STARPU_HETEROPRIO_NOD_TIME_COMBINATION , **STARPU_HETEROPRIO_BEST_NODS_SCORE** , **STARPU_HETEROPRIO_BEST_NODS** , **STARPU_HETEROPRIO_URT_PURE** ,
  STARPU_HETEROPRIO_URT , **STARPU_HETEROPRIO_URT_2** , **STARPU_HETEROPRIO_URT_DOT_DIFF_PURE** , **STARPU_HETEROPRIO_URT_DOT_DIFF_PURE_2** ,
  STARPU_HETEROPRIO_URT_DOT_REL_DIFF_PURE , **STARPU_HETEROPRIO_URT_DOT_REL_DIFF_PURE_2** , **STARPU_HETEROPRIO_URT_DOT_DIFF_2** , **STARPU_HETEROPRIO_URT_DOT_DIFF_3** ,
  STARPU_HETEROPRIO_URT_DOT_DIFF_4 , **STARPU_HETEROPRIO_URT_DOT_DIFF_5** , **STARPU_HETEROPRIO_URT_DOT_DIFF_6** , **STARPU_HETEROPRIO_URT_DOT_DIFF_7** ,
  STARPU_HETEROPRIO_URT_DOT_DIFF_8 , **STARPU_HETEROPRIO_URT_DOT_DIFF_9** , **STARPU_HETEROPRIO_URT_DOT_DIFF_10** , **STARPU_HETEROPRIO_URT_DOT_DIFF_11** ,
  STARPU_HETEROPRIO_URTS_PER_SECONDS , **STARPU_HETEROPRIO_URTS_PER_SECONDS_2** , **STARPU_HETEROPRIO_URTS_PER_SECONDS_DIFF** , **STARPU_HETEROPRIO_URTS_TIME_RELEASED_DIFF** ,
  STARPU_HETEROPRIO_URTS_TIME_COMBINATION , **STARPU_HETEROPRIO_NODS_PER_SECOND** , **STARPU_HETEROPRIO_NODS_TIME_RELEASED** , **STARPU_HETEROPRIO_NODS_TIME_RELEASED_DIFF** }

Functions

- void starpu_heteroprio_set_use_locality (unsigned sched_ctx_id, unsigned use_locality)
- void starpu_heteroprio_set_nb_prios (unsigned sched_ctx_id, enum starpu_worker_archtype arch, unsigned max_prio)
- void starpu_heteroprio_set_mapping (unsigned sched_ctx_id, enum starpu_worker_archtype arch, unsigned source_prio, unsigned dest_bucket_id)
- void starpu_heteroprio_set_faster_arch (unsigned sched_ctx_id, enum starpu_worker_archtype arch, unsigned bucket_id)
- void starpu_heteroprio_set_arch_slow_factor (unsigned sched_ctx_id, enum starpu_worker_archtype arch, unsigned bucket_id, float slow_factor)
- void starpu_heteroprio_map_wgroup_memory_nodes (unsigned sched_ctx_id)
- void starpu_heteroprio_print_wgroups (FILE ∗stream, unsigned sched_ctx_id)

Variables

- static const char **starpu_autoheteroprio_priority_ordering_policy_names** [STARPU_AUTOHETEROPRIO_PRIORITY_ORDERING_POLICY_COUNT][64]

57.13.1 Detailed Description

This is the interface for the heteroprio scheduler.

57.13.2 Enumeration Type Documentation

57.13.2.1 starpu_autoheteroprio_priority_ordering_policy

```
enum starpu_autoheteroprio_priority_ordering_policy
```

todo

57.13.3 Function Documentation

57.13.3.1 starpu_heteroprio_set_use_locality()

```
void starpu_heteroprio_set_use_locality (
            unsigned sched_ctx_id,
            unsigned use_locality )
```

Set if heteroprio should use data locality or not

57.13.3.2 starpu_heteroprio_set_nb_prios()

```
void starpu_heteroprio_set_nb_prios (
            unsigned sched_ctx_id,
            enum starpu_worker_archtype arch,
            unsigned max_prio )
```

Tell how many prio there are for a given arch

57.13.3.3 starpu_heteroprio_set_mapping()

```
void starpu_heteroprio_set_mapping (
            unsigned sched_ctx_id,
            enum starpu_worker_archtype arch,
            unsigned source_prio,
            unsigned dest_bucket_id )
```

Set the mapping for a given arch prio=>bucket

57.13.3.4 starpu_heteroprio_set_faster_arch()

```
void starpu_heteroprio_set_faster_arch (
            unsigned sched_ctx_id,
            enum starpu_worker_archtype arch,
            unsigned bucket_id )
```

Tell which arch is the faster for the tasks of a bucket (optional)

57.13.3.5 starpu_heteroprio_set_arch_slow_factor()

```
void starpu_heteroprio_set_arch_slow_factor (
            unsigned sched_ctx_id,
            enum starpu_worker_archtype arch,
            unsigned bucket_id,
            float slow_factor )
```

Tell how slow is a arch for the tasks of a bucket (optional)

57.13.3.6 starpu_heteroprio_map_wgroup_memory_nodes()

```
void starpu_heteroprio_map_wgroup_memory_nodes (
            unsigned sched_ctx_id )
```

One memory node will be one wgroup

57.13.3.7 starpu_heteroprio_print_wgroups()

```
void starpu_heteroprio_print_wgroups (
            FILE * stream,
            unsigned sched_ctx_id )
```

Print the current setup groups

57.14 HIP Extensions

Macros

- #define STARPU_USE_HIP
- #define STARPU_MAXHIPDEVS
- #define STARPU_HIPBLAS_REPORT_ERROR(status)
- #define STARPU_HIP_REPORT_ERROR(status)

Functions

- void starpu_hipblas_report_error (const char ∗func, const char ∗file, int line, int status)
- void starpu_hip_report_error (const char ∗func, const char ∗file, int line, hipError_t status)
- hipStream_t starpu_hip_get_local_stream (void)
- const struct hipDeviceProp_t ∗ starpu_hip_get_device_properties (unsigned workerid)
- int starpu_hip_copy_async_sync (void ∗src_ptr, unsigned src_node, void ∗dst_ptr, unsigned dst_node, size_t ssize, hipStream_t stream, hipMemcpyKind kind)
- int starpu_hip_copy2d_async_sync (void ∗src_ptr, unsigned src_node, void ∗dst_ptr, unsigned dst_node, size_t blocksize, size_t numblocks, size_t ld_src, size_t ld_dst, hipStream_t stream, hipMemcpyKind kind)
- int starpu_hip_copy3d_async_sync (void ∗src_ptr, unsigned src_node, void ∗dst_ptr, unsigned dst_node, size_t blocksize, size_t numblocks_1, size_t ld1_src, size_t ld1_dst, size_t numblocks_2, size_t ld2_src, size_t ld2_dst, hipStream_t stream, hipMemcpyKind kind)
- void starpu_hip_set_device (int devid)

57.14.1 Detailed Description

57.14.2 Macro Definition Documentation

57.14.2.1 STARPU_USE_HIP

`#define STARPU_USE_HIP`

Defined when StarPU has been installed with HIP support. It should be used in your code to detect the availability of HIP.

57.14.2.2 STARPU_MAXHIPDEVS

`#define STARPU_MAXHIPDEVS`

Define the maximum number of HIP devices that are supported by StarPU.

57.14.2.3 STARPU_HIPBLAS_REPORT_ERROR

```
#define STARPU_HIPBLAS_REPORT_ERROR(
              status )
```

Call starpu_hipblas_report_error(), passing the current function, file and line position.

57.14.2.4 STARPU_HIP_REPORT_ERROR

```
#define STARPU_HIP_REPORT_ERROR(
              status )
```

Call starpu_hip_report_error(), passing the current function, file and line position.

57.14.3 Function Documentation

57.14.3.1 starpu_hipblas_report_error()

```
void starpu_hipblas_report_error (
            const char * func,
            const char * file,
            int line,
            int status )
```

Report a HIPBLAS error.

57.14.3.2 starpu_hip_report_error()

```
void starpu_hip_report_error (
            const char * func,
            const char * file,
            int line,
            hipError_t status )
```

Report a HIP error.

57.14.3.3 starpu_hip_get_local_stream()

```
hipStream_t starpu_hip_get_local_stream (
            void )
```

Return the current worker's HIP stream. StarPU provides a stream for every HIP device controlled by StarPU. This function is only provided for convenience so that programmers can easily use asynchronous operations within codelets without having to create a stream by hand. Note that the application is not forced to use the stream provided by starpu_hip_get_local_stream() and may also create its own streams. Synchronizing with `hipDeviceSynchronize()` is allowed, but will reduce the likelihood of having all transfers overlapped.

57.14.3.4 starpu_hip_get_device_properties()

```
const struct hipDeviceProp_t * starpu_hip_get_device_properties (
            unsigned workerid )
```

Return a pointer to device properties for worker `workerid` (assumed to be a HIP worker).

57.14.3.5 starpu_hip_copy_async_sync()

```
int starpu_hip_copy_async_sync (
            void * src_ptr,
            unsigned src_node,
            void * dst_ptr,
            unsigned dst_node,
            size_t ssize,
            hipStream_t stream,
            hipMemcpyKind kind )
```

Copy `ssize` bytes from the pointer `src_ptr` on `src_node` to the pointer `dst_ptr` on `dst_node`. The function first tries to copy the data asynchronous (unless `stream` is `NULL`). If the asynchronous copy fails or if `stream` is `NULL`, it copies the data synchronously. The function returns `-EAGAIN` if the asynchronous launch was successful. It returns 0 if the synchronous copy was successful, or fails otherwise.

57.14.3.6 starpu_hip_copy2d_async_sync()

```
int starpu_hip_copy2d_async_sync (
            void * src_ptr,
            unsigned src_node,
            void * dst_ptr,
            unsigned dst_node,
            size_t blocksize,
            size_t numblocks,
            size_t ld_src,
            size_t ld_dst,
            hipStream_t stream,
            hipMemcpyKind kind )
```

Copy `numblocks` blocks of `blocksize` bytes from the pointer `src_ptr` on `src_node` to the pointer `dst_ptr` on `dst_node`.

The blocks start at addresses which are ld_src (resp. ld_dst) bytes apart in the source (resp. destination) interface. The function first tries to copy the data asynchronous (unless `stream` is `NULL`). If the asynchronous copy fails or if `stream` is `NULL`, it copies the data synchronously. The function returns `-EAGAIN` if the asynchronous launch was successful. It returns 0 if the synchronous copy was successful, or fails otherwise.

57.14.3.7 starpu_hip_copy3d_async_sync()

```
int starpu_hip_copy3d_async_sync (
            void * src_ptr,
            unsigned src_node,
            void * dst_ptr,
            unsigned dst_node,
            size_t blocksize,
            size_t numblocks_1,
            size_t ld1_src,
            size_t ld1_dst,
            size_t numblocks_2,
            size_t ld2_src,
            size_t ld2_dst,
            hipStream_t stream,
            hipMemcpyKind kind )
```

Copy `numblocks_1 * numblocks_2` blocks of `blocksize` bytes from the pointer `src_ptr` on `src_node` to the pointer `dst_ptr` on `dst_node`.

The blocks are grouped by `numblocks_1` blocks whose start addresses are ld1_src (resp. ld1_dst) bytes apart in the source (resp. destination) interface.

The function first tries to copy the data asynchronous (unless `stream` is `NULL`). If the asynchronous copy fails or if `stream` is `NULL`, it copies the data synchronously. The function returns `-EAGAIN` if the asynchronous launch was successful. It returns 0 if the synchronous copy was successful, or fails otherwise.

57.14.3.8 starpu_hip_set_device()

```
void starpu_hip_set_device (
            int devid )
```

Call `hipSetDevice(devid)`.

57.15 Initialization and Termination

Data Structures

- struct starpu_conf

Macros

- #define STARPU_THREAD_ACTIVE

Functions

- int starpu_conf_init (struct starpu_conf ∗conf)
- int starpu_conf_noworker (struct starpu_conf ∗conf)
- int starpu_init (struct starpu_conf ∗conf)
- int starpu_initialize (struct starpu_conf ∗user_conf, int ∗argc, char ∗∗∗argv)
- int starpu_is_initialized (void)
- void starpu_wait_initialized (void)
- void starpu_shutdown (void)
- void starpu_pause (void)
- void starpu_resume (void)
- int starpu_is_paused (void)
- unsigned starpu_get_next_bindid (unsigned flags, unsigned ∗preferred, unsigned npreferred)
- int starpu_bind_thread_on (int cpuid, unsigned flags, const char ∗name)
- void starpu_bind_thread_on_worker (unsigned workerid)
- void starpu_bind_thread_on_main (void)
- void starpu_bind_thread_on_cpu (int cpuid)
- int starpu_cpu_os_index (int cpuid)
- void starpu_topology_print (FILE ∗f)
- int starpu_asynchronous_copy_disabled (void)
- int starpu_asynchronous_cuda_copy_disabled (void)
- int starpu_asynchronous_hip_copy_disabled (void)
- int starpu_asynchronous_opencl_copy_disabled (void)
- int starpu_asynchronous_max_fpga_copy_disabled (void)
- int starpu_asynchronous_mpi_ms_copy_disabled (void)
- int starpu_asynchronous_tcpip_ms_copy_disabled (void)
- int starpu_asynchronous_copy_disabled_for (enum starpu_node_kind kind)
- int starpu_map_enabled (void)
- void starpu_display_stats (void)

57.15.1 Detailed Description

57.15.2 Data Structure Documentation

57.15.2.1 struct starpu_conf

Structure passed to the starpu_init() function to configure StarPU. It has to be initialized with starpu_conf_init(). When the default value is used, StarPU automatically selects the number of processing units and takes the default scheduling policy. The environment variables overwrite the equivalent parameters unless starpu_conf::precedence_over_environment_variables is set.

Data Fields

- const char ∗ sched_policy_name
- struct starpu_sched_policy ∗ sched_policy
- void(∗ sched_policy_callback)(unsigned)
- int precedence_over_environment_variables
- int ncpus
- int reserve_ncpus
- int ncuda
- int nhip
- int nopencl
- int nmax_fpga
- int nmpi_ms
- int ntcpip_ms
- unsigned use_explicit_workers_bindid
- unsigned workers_bindid [STARPU_NMAXWORKERS]
- unsigned use_explicit_workers_cuda_gpuid
- unsigned workers_cuda_gpuid [STARPU_NMAXWORKERS]
- unsigned use_explicit_workers_hip_gpuid
- unsigned workers_hip_gpuid [STARPU_NMAXWORKERS]
- unsigned use_explicit_workers_opencl_gpuid
- unsigned workers_opencl_gpuid [STARPU_NMAXWORKERS]
- unsigned use_explicit_workers_max_fpga_deviceid
- unsigned workers_max_fpga_deviceid [STARPU_NMAXWORKERS]
- struct starpu_max_load ∗ max_fpga_load
- unsigned use_explicit_workers_mpi_ms_deviceid
- unsigned workers_mpi_ms_deviceid [STARPU_NMAXWORKERS]
- int bus_calibrate
- int calibrate
- int data_locality_enforce
- int single_combined_worker
- int disable_asynchronous_copy
- int disable_asynchronous_cuda_copy
- int disable_asynchronous_hip_copy
- int disable_asynchronous_opencl_copy
- int disable_asynchronous_mpi_ms_copy
- int disable_asynchronous_tcpip_ms_copy
- int disable_asynchronous_max_fpga_copy
- int enable_map
- unsigned ∗ cuda_opengl_interoperability
- unsigned n_cuda_opengl_interoperability
- struct starpu_driver ∗ not_launched_drivers
- unsigned n_not_launched_drivers
- uint64_t trace_buffer_size
- int global_sched_ctx_min_priority
- int global_sched_ctx_max_priority
- void(∗ **callback_worker_going_to_sleep**)(unsigned workerid)
- void(∗ **callback_worker_waking_up**)(unsigned workerid)
- int catch_signals
- unsigned start_perf_counter_collection
- unsigned driver_spinning_backoff_min
- unsigned driver_spinning_backoff_max
- int cuda_only_fast_alloc_other_memnodes

Private Attributes

- int magic
- int will_use_mpi

57.15.2.1.1 Field Documentation

57.15.2.1.1.1 magic `int starpu_conf::magic [private]`

Will be initialized by starpu_conf_init(). Should not be set by hand.

57.15.2.1.1.2 will_use_mpi `int starpu_conf::will_use_mpi [private]`

Tell starpu_init() if MPI will be initialized later.

57.15.2.1.1.3 sched_policy_name `const char* starpu_conf::sched_policy_name`

Name of the scheduling policy. This can also be specified with the environment variable STARPU_SCHED. (default = `NULL`).

57.15.2.1.1.4 sched_policy `struct starpu_sched_policy* starpu_conf::sched_policy`

Definition of the scheduling policy. This field is ignored if starpu_conf::sched_policy_name is set. (default = `NULL`)

57.15.2.1.1.5 sched_policy_callback `void(* starpu_conf::sched_policy_callback) (unsigned)`

Callback function that can later be used by the scheduler. The scheduler can retrieve this function by calling starpu_sched_ctx_get_sched_policy_callback()

57.15.2.1.1.6 precedence_over_environment_variables `int starpu_conf::precedence_over_environment_variables`

For all parameters specified in this structure that can also be set with environment variables, by default, StarPU chooses the value of the environment variable against the value set in starpu_conf. Setting the parameter starpu_conf::precedence_over_environment_variables to 1 allows to give precedence to the value set in the structure over the environment variable.

57.15.2.1.1.7 ncpus `int starpu_conf::ncpus`

Number of CPU cores that StarPU can use. This can also be specified with the environment variable STARPU_NCPU. (default = `-1`)

57.15.2.1.1.8 reserve_ncpus `int starpu_conf::reserve_ncpus`

Number of CPU cores to that StarPU should leave aside. They can then be used by application threads, by calling starpu_get_next_bindid() to get their ID, and starpu_bind_thread_on() to bind the current thread to them.

57.15.2.1.1.9 ncuda `int starpu_conf::ncuda`

Number of CUDA devices that StarPU can use. This can also be specified with the environment variable STARPU_NCUDA. (default = `-1`)

57.15.2.1.1.10 nhip `int starpu_conf::nhip`

Number of HIP devices that StarPU can use. This can also be specified with the environment variable STARPU_NHIP. (default = `-1`)

57.15.2.1.1.11 nopencl `int starpu_conf::nopencl`

Number of OpenCL devices that StarPU can use. This can also be specified with the environment variable STARPU_NOPENCL. (default = `-1`)

57.15.2.1.1.12 nmax_fpga `int starpu_conf::nmax_fpga`

Number of Maxeler FPGA devices that StarPU can use. This can also be specified with the environment variable STARPU_NMAX_FPGA. (default = -1)

57.15.2.1.1.13 nmpi_ms `int starpu_conf::nmpi_ms`

Number of MPI Master Slave devices that StarPU can use. This can also be specified with the environment variable STARPU_NMPI_MS. (default = -1)

57.15.2.1.1.14 ntcpip_ms `int starpu_conf::ntcpip_ms`

Number of TCP/IP Master Slave devices that StarPU can use. This can also be specified with the environment variable STARPU_NTCPIP_MS. (default = -1)

57.15.2.1.1.15 use_explicit_workers_bindid `unsigned starpu_conf::use_explicit_workers_bindid`

If this flag is set, the starpu_conf::workers_bindid array indicates where the different workers are bound, otherwise StarPU automatically selects where to bind the different workers. This can also be specified with the environment variable STARPU_WORKERS_CPUID. (default = 0)

57.15.2.1.1.16 workers_bindid `unsigned starpu_conf::workers_bindid[STARPU_NMAXWORKERS]`

If the starpu_conf::use_explicit_workers_bindid flag is set, this array indicates where to bind the different workers. The i-th entry of the starpu_conf::workers_bindid indicates the logical identifier of the processor which should execute the i-th worker. Note that the logical ordering of the CPUs is either determined by the OS, or provided by the `hwloc` library in case it is available.

57.15.2.1.1.17 use_explicit_workers_cuda_gpuid `unsigned starpu_conf::use_explicit_workers_cuda↩ _gpuid`

If this flag is set, the CUDA workers will be attached to the CUDA devices specified in the starpu_conf::workers_cuda_gpuid array. Otherwise, StarPU affects the CUDA devices in a round-robin fashion. This can also be specified with the environment variable STARPU_WORKERS_CUDAID. (default = 0)

57.15.2.1.1.18 workers_cuda_gpuid `unsigned starpu_conf::workers_cuda_gpuid[STARPU_NMAXWORKERS]`

If the starpu_conf::use_explicit_workers_cuda_gpuid flag is set, this array contains the logical identifiers of the CUDA devices (as used by `cudaGetDevice()`).

57.15.2.1.1.19 use_explicit_workers_hip_gpuid `unsigned starpu_conf::use_explicit_workers_hip_↩ gpuid`

If this flag is set, the HIP workers will be attached to the HIP devices specified in the starpu_conf::workers_hip_gpuid array. Otherwise, StarPU affects the HIP devices in a round-robin fashion. This can also be specified with the environment variable STARPU_WORKERS_HIPID. (default = 0)

57.15.2.1.1.20 workers_hip_gpuid `unsigned starpu_conf::workers_hip_gpuid[STARPU_NMAXWORKERS]`

If the starpu_conf::use_explicit_workers_hip_gpuid flag is set, this array contains the logical identifiers of the HIP devices (as used by `hipGetDevice()`).

57.15.2.1.1.21 use_explicit_workers_opencl_gpuid `unsigned starpu_conf::use_explicit_workers_↩ opencl_gpuid`

If this flag is set, the OpenCL workers will be attached to the OpenCL devices specified in the starpu_conf::workers_opencl_gpuid array. Otherwise, StarPU affects the OpenCL devices in a round-robin fashion. This can also be specified with the environment variable STARPU_WORKERS_OPENCLID. (default = 0)

57.15.2.1.1.22 workers_opencl_gpuid `unsigned starpu_conf::workers_opencl_gpuid[STARPU_NMAXWORKERS]`

If the starpu_conf::use_explicit_workers_opencl_gpuid flag is set, this array contains the logical identifiers of the OpenCL devices to be used.

57.15.2.1.1.23 use_explicit_workers_max_fpga_deviceid `unsigned starpu_conf::use_explicit_↩ workers_max_fpga_deviceid`

If this flag is set, the Maxeler FPGA workers will be attached to the Maxeler FPGA devices specified in the starpu_conf::workers_max_fpga_deviceid array. Otherwise, StarPU affects the Maxeler FPGA devices in a round-robin fashion. This can also be specified with the environment variable STARPU_WORKERS_MAX_FPGAID. (default = 0)

57.15.2.1.1.24 workers_max_fpga_deviceid `unsigned starpu_conf::workers_max_fpga_deviceid[STARPU_NMAXWORKERS]`

If the starpu_conf::use_explicit_workers_max_fpga_deviceid flag is set, this array contains the logical identifiers of the Maxeler FPGA devices to be used.

57.15.2.1.1.25 max_fpga_load `struct starpu_max_load* starpu_conf::max_fpga_load`

This allows to specify the Maxeler file(s) to be loaded on Maxeler FPGAs. This is an array of starpu_max_load, the last of which shall have file set to NULL. In order to use all available devices, starpu_max_load::engine_id_pattern can be set to "*", but only the last non-NULL entry can be set so.

If this is not set, it is assumed that the basic static SLiC interface is used.

57.15.2.1.1.26 use_explicit_workers_mpi_ms_deviceid `unsigned starpu_conf::use_explicit_workers_mpi_ms_deviceid`

If this flag is set, the MPI Master Slave workers will be attached to the MPI Master Slave devices specified in the array starpu_conf::workers_mpi_ms_deviceid. Otherwise, StarPU affects the MPI Master Slave devices in a round-robin fashion. (default = 0)

57.15.2.1.1.27 workers_mpi_ms_deviceid `unsigned starpu_conf::workers_mpi_ms_deviceid[STARPU_NMAXWORKERS]`

If the flag starpu_conf::use_explicit_workers_mpi_ms_deviceid is set, the array contains the logical identifiers of the MPI Master Slave devices to be used.

57.15.2.1.1.28 bus_calibrate `int starpu_conf::bus_calibrate`

If this flag is set, StarPU will recalibrate the bus. If this value is equal to -1, the default value is used. This can also be specified with the environment variable STARPU_BUS_CALIBRATE. (default = 0)

57.15.2.1.1.29 calibrate `int starpu_conf::calibrate`

If this flag is set, StarPU will calibrate the performance models when executing tasks. If this value is equal to -1, the default value is used. If the value is equal to 1, it will force continuing calibration. If the value is equal to 2, the existing performance models will be overwritten. This can also be specified with the environment variable STARPU_CALIBRATE. (default = 0)

57.15.2.1.1.30 data_locality_enforce `int starpu_conf::data_locality_enforce`

This flag should be set to 1 to enforce data locality when choosing a worker to execute a task. This can also be specified with the environment variable STARPU_DATA_LOCALITY_ENFORCE. This can also be specified at compilation time by giving to the configure script the option --enable-data-locality-enforce. (default = 0)

57.15.2.1.1.31 single_combined_worker `int starpu_conf::single_combined_worker`

By default, StarPU executes parallel tasks concurrently. Some parallel libraries (e.g. most OpenMP implementations) however do not support concurrent calls to parallel code. In such case, setting this flag makes StarPU only start one parallel task at a time (but other CPU and GPU tasks are not affected and can be run concurrently). The parallel task scheduler will however still try varying combined worker sizes to look for the most efficient ones. This can also be specified with the environment variable STARPU_SINGLE_COMBINED_WORKER. (default = 0)

57.15.2.1.1.32 disable_asynchronous_copy `int starpu_conf::disable_asynchronous_copy`

This flag should be set to 1 to disable asynchronous copies between CPUs and all accelerators. The AMD implementation of OpenCL is known to fail when copying data asynchronously. When using this implementation, it is therefore necessary to disable asynchronous data transfers. This can also be specified with the environment variable STARPU_DISABLE_ASYNCHRONOUS_COPY. This can also be specified at compilation time by giving to the configure script the option --disable-asynchronous-copy. (default = 0)

57.15.2.1.1.33 disable_asynchronous_cuda_copy `int starpu_conf::disable_asynchronous_cuda_copy`

This flag should be set to 1 to disable asynchronous copies between CPUs and CUDA accelerators. This can also be specified with the environment variable STARPU_DISABLE_ASYNCHRONOUS_CUDA_COPY. This can also be specified at compilation time by giving to the configure script the option --disable-asynchronous-cuda-copy. (default = 0)

57.15.2.1.1.34 disable_asynchronous_hip_copy `int starpu_conf::disable_asynchronous_hip_copy`

This flag should be set to 1 to disable asynchronous copies between CPUs and HIP accelerators. This can also be specified with the environment variable STARPU_DISABLE_ASYNCHRONOUS_HIP_COPY. This can also be specified at compilation time by giving to the configure script the option --disable-asynchronous-hip-copy. (default = `0`)

57.15.2.1.1.35 disable_asynchronous_opencl_copy `int starpu_conf::disable_asynchronous_opencl_copy`

This flag should be set to 1 to disable asynchronous copies between CPUs and OpenCL accelerators. The AMD implementation of OpenCL is known to fail when copying data asynchronously. When using this implementation, it is therefore necessary to disable asynchronous data transfers. This can also be specified with the environment variable STARPU_DISABLE_ASYNCHRONOUS_OPENCL_COPY. This can also be specified at compilation time by giving to the configure script the option --disable-asynchronous-opencl-copy. (default = `0`)

57.15.2.1.1.36 disable_asynchronous_mpi_ms_copy `int starpu_conf::disable_asynchronous_mpi_ms_copy`

This flag should be set to 1 to disable asynchronous copies between CPUs and MPI Master Slave devices. This can also be specified with the environment variable STARPU_DISABLE_ASYNCHRONOUS_MPI_MS_COPY. This can also be specified at compilation time by giving to the configure script the option --disable-asynchronous-mpi-master-slave-copy. (default = `0`).

57.15.2.1.1.37 disable_asynchronous_tcpip_ms_copy `int starpu_conf::disable_asynchronous_tcpip_ms_copy`

This flag should be set to 1 to disable asynchronous copies between CPUs and TCP/IP Master Slave devices. This can also be specified with the environment variable STARPU_DISABLE_ASYNCHRONOUS_TCPIP_MS_COPY. This can also be specified at compilation time by giving to the configure script the option --disable-asynchronous-tcpip-master-slave-cop
(default = `0`).

57.15.2.1.1.38 disable_asynchronous_max_fpga_copy `int starpu_conf::disable_asynchronous_max_fpga_copy`

This flag should be set to 1 to disable asynchronous copies between CPUs and Maxeler FPGA devices. This can also be specified with the environment variable STARPU_DISABLE_ASYNCHRONOUS_MAX_FPGA_COPY. This can also be specified at compilation time by giving to the configure script the option --disable-asynchronous-fpga-copy. (default = 0).

57.15.2.1.1.39 enable_map `int starpu_conf::enable_map`

This flag should be set to 1 to disable memory mapping support between memory nodes. This can also be specified with the environment variable STARPU_ENABLE_MAP.

57.15.2.1.1.40 cuda_opengl_interoperability `unsigned* starpu_conf::cuda_opengl_interoperability`

Enable CUDA/OpenGL interoperation on these CUDA devices. This can be set to an array of CUDA device identifiers for which `cudaGLSetGLDevice()` should be called instead of `cudaSetDevice()`. Its size is specified by the starpu_conf::n_cuda_opengl_interoperability field below (default = `NULL`)

57.15.2.1.1.41 n_cuda_opengl_interoperability `unsigned starpu_conf::n_cuda_opengl_interoperability`

Size of the array starpu_conf::cuda_opengl_interoperability

57.15.2.1.1.42 not_launched_drivers `struct starpu_driver* starpu_conf::not_launched_drivers`

Array of drivers that should not be launched by StarPU. The application will run in one of its own threads. (default = `NULL`)

57.15.2.1.1.43 n_not_launched_drivers `unsigned starpu_conf::n_not_launched_drivers`

The number of StarPU drivers that should not be launched by StarPU, i.e number of elements of the array starpu_conf::not_launched_drivers. (default = `0`)

57.15.2.1.1.44 trace_buffer_size `uint64_t starpu_conf::trace_buffer_size`

Specify the buffer size used for FxT tracing. Starting from FxT version 0.2.12, the buffer will automatically be flushed when it fills in, but it may still be interesting to specify a bigger value to avoid any flushing (which would disturb the trace).

57.15.2.1.1.45 global_sched_ctx_min_priority `int starpu_conf::global_sched_ctx_min_priority`

Set the minimum priority used by priorities-aware schedulers. This also can be specified with the environment variable STARPU_MIN_PRIO

57.15.2.1.1.46 global_sched_ctx_max_priority `int starpu_conf::global_sched_ctx_max_priority`

Set the maximum priority used by priorities-aware schedulers. This also can be specified with the environment variable STARPU_MAX_PRIO

57.15.2.1.1.47 catch_signals `int starpu_conf::catch_signals`

Specify if StarPU should catch `SIGINT`, `SIGSEGV` and `SIGTRAP` signals to make sure final actions (e.g dumping FxT trace files) are done even though the application has crashed. By default (value = `1`), signals are caught. It should be disabled on systems which already catch these signals for their own needs (e.g JVM) This can also be specified with the environment variable STARPU_CATCH_SIGNALS.

57.15.2.1.1.48 start_perf_counter_collection `unsigned starpu_conf::start_perf_counter_collection`

Specify whether StarPU should automatically start to collect performance counters after initialization

57.15.2.1.1.49 driver_spinning_backoff_min `unsigned starpu_conf::driver_spinning_backoff_min`

Minimum spinning backoff of drivers (default = `1`)

57.15.2.1.1.50 driver_spinning_backoff_max `unsigned starpu_conf::driver_spinning_backoff_max`

Maximum spinning backoff of drivers. (default = `32`)

57.15.2.1.1.51 cuda_only_fast_alloc_other_memnodes `int starpu_conf::cuda_only_fast_alloc_other_memnodes`

Specify if CUDA workers should do only fast allocations when running the datawizard progress of other memory nodes. This will pass the interval value _STARPU_DATAWIZARD_ONLY_FAST_ALLOC to the allocation method. Default value is 0, allowing CUDA workers to do slow allocations. This can also be specified with the environment variable STARPU_CUDA_ONLY_FAST_ALLOC_OTHER_MEMNODES.

57.15.3 Macro Definition Documentation

57.15.3.1 STARPU_THREAD_ACTIVE

```
#define STARPU_THREAD_ACTIVE
```

Value to be passed to starpu_get_next_bindid() and starpu_bind_thread_on() when binding a thread which will significantly eat CPU time, and should thus have its own dedicated CPU.

57.15.4 Function Documentation

57.15.4.1 starpu_conf_init()

```
int starpu_conf_init (
            struct starpu_conf * conf )
```

Initialize the `conf` structure with the default values. In case some configuration parameters are already specified through environment variables, starpu_conf_init() initializes the fields of `conf` according to the environment variables. For instance if STARPU_CALIBRATE is set, its value is put in the field starpu_conf::calibrate of `conf`. Upon successful completion, this function returns 0. Otherwise, `-EINVAL` indicates that the argument was `NULL`.

57.15.4.2 starpu_conf_noworker()

```
int starpu_conf_noworker (
            struct starpu_conf * conf )
```

Set fields of `conf` so that no worker is enabled, i.e. set starpu_conf::ncpus = 0, starpu_conf::ncuda = 0, etc.
This allows to portably enable only a given type of worker:

```
starpu_conf_noworker(&conf);
conf.ncpus = -1;
```

See Configuration and Initialization for more details.

57.15.4.3 starpu_init()

```
int starpu_init (
            struct starpu_conf * conf )
```

StarPU initialization method, must be called prior to any other StarPU call. It is possible to specify StarPU's configuration (e.g. scheduling policy, number of cores, ...) by passing a non-`NULL` `conf`. Default configuration is used if `conf` is `NULL`. Upon successful completion, this function returns 0. Otherwise, `-ENODEV` indicates that no worker was available (and thus StarPU was not initialized). See Submitting A Task for more details.

57.15.4.4 starpu_initialize()

```
int starpu_initialize (
            struct starpu_conf * user_conf,
            int * argc,
            char *** argv )
```

Similar to starpu_init(), but also take the `argc` and `argv` as defined by the application, which is necessary when running in Simgrid mode or MPI Master Slave mode. Do not call starpu_init() and starpu_initialize() in the same program. See Submitting A Task for more details.

57.15.4.5 starpu_is_initialized()

```
int starpu_is_initialized (
            void )
```

Return 1 if StarPU is already initialized. See Configuration and Initialization for more details.

57.15.4.6 starpu_wait_initialized()

```
void starpu_wait_initialized (
            void )
```

Wait for starpu_init() call to finish. See Configuration and Initialization for more details.

57.15.4.7 starpu_shutdown()

```
void starpu_shutdown (
            void )
```

StarPU termination method, must be called at the end of the application: statistics and other post-mortem debugging information are not guaranteed to be available until this method has been called. See Submitting A Task for more details.

57.15.4.8 starpu_pause()

```
void starpu_pause (
            void )
```

Suspend the processing of new tasks by workers. It can be used in a program where StarPU is used during only a part of the execution. Without this call, the workers continue to poll for new tasks in a tight loop, wasting CPU time. The symmetric call to starpu_resume() should be used to unfreeze the workers. See Kernel Threads Started by StarPU and Interleaving StarPU and non-StarPU code for more details.

57.15.4.9 starpu_resume()

```
void starpu_resume (
            void )
```

Symmetrical call to starpu_pause(), used to resume the workers polling for new tasks. This would be typically called only once having submitted all tasks. See Kernel Threads Started by StarPU and Interleaving StarPU and non-StarPU code for more details.

57.15.4.10 starpu_is_paused()

```
int starpu_is_paused (
            void )
```

Return !0 if task processing by workers is currently paused, 0 otherwise. See StarPU permanently eats 100% of all CPUs for more details.

57.15.4.11 starpu_get_next_bindid()

```
unsigned starpu_get_next_bindid (
            unsigned flags,
            unsigned * preferred,
            unsigned npreferred )
```

Return a PU binding ID which can be used to bind threads with starpu_bind_thread_on(). `flags` can be set to STARPU_THREAD_ACTIVE or 0. When `npreferred` is set to non-zero, `preferred` is an array of size `npreferred` in which a preference of PU binding IDs can be set. By default StarPU will return the first PU available for binding. See Kernel Threads Started by StarPU and CPU Workers for more details.

57.15.4.12 starpu_bind_thread_on()

```
int starpu_bind_thread_on (
            int cpuid,
            unsigned flags,
            const char * name )
```

Bind the calling thread on the given `cpuid` (which should have been obtained with starpu_get_next_bindid()). Return -1 if a thread was already bound to this PU (but binding will still have been done, and a warning will have been printed), so the caller can tell the user how to avoid the issue.
`name` should be set to a unique string so that different calls with the same name for the same `cpuid` does not produce a warning.
See Kernel Threads Started by StarPU and CPU Workers for more details.

57.15.4.13 starpu_bind_thread_on_worker()

```
void starpu_bind_thread_on_worker (
            unsigned workerid )
```

Bind the calling thread on the cores corresponding to the `workerid` .
`workerid` can be a basic worker or a combined worker.
This can be used e.g. before initializing a library which records at initialization time the thread binding to be used when running kernels.
See Kernel Threads Started by StarPU and CPU Workers for more details.

57.15.4.14 starpu_bind_thread_on_main()

```
void starpu_bind_thread_on_main (
            void )
```

Bind the calling thread back to the core reserved for the main thread.
This can be used e.g. after initializing a library which records at initialization time the thread binding to be used when running kernels.
See Kernel Threads Started by StarPU and CPU Workers for more details.

57.15.4.15 starpu_bind_thread_on_cpu()

```
void starpu_bind_thread_on_cpu (
            int cpuid )
```

Bind the calling thread on the given `cpuid`

This can be used e.g. after initializing a library which records at initialization time the thread binding to be used when running kernels.

See Kernel Threads Started by StarPU and CPU Workers for more details.

57.15.4.16 starpu_cpu_os_index()

```
int starpu_cpu_os_index (
            int cpuid )
```

Return the OS number of a given `cpuid`

StarPU uses logical numbering (as define by hwloc) all along, but in case interaction is needed with another binding tool that uses numbering as defined by the OS, we need to convert from hwloc logical numbering to hwloc physical numbering.

57.15.4.17 starpu_topology_print()

```
void starpu_topology_print (
            FILE * f )
```

Print a description of the topology on `f`. See Configuration and Initialization for more details.

57.15.4.18 starpu_asynchronous_copy_disabled()

```
int starpu_asynchronous_copy_disabled (
            void )
```

Return 1 if asynchronous data transfers between CPU and accelerators are disabled. See General Configuration for more details.

57.15.4.19 starpu_asynchronous_cuda_copy_disabled()

```
int starpu_asynchronous_cuda_copy_disabled (
            void )
```

Return 1 if asynchronous data transfers between CPU and CUDA accelerators are disabled. See CUDA Workers for more details.

57.15.4.20 starpu_asynchronous_hip_copy_disabled()

```
int starpu_asynchronous_hip_copy_disabled (
            void )
```

Return 1 if asynchronous data transfers between CPU and HIP accelerators are disabled. See HIP Workers for more details.

57.15.4.21 starpu_asynchronous_opencl_copy_disabled()

```
int starpu_asynchronous_opencl_copy_disabled (
            void )
```

Return 1 if asynchronous data transfers between CPU and OpenCL accelerators are disabled. See OpenCL Workers for more details.

57.15.4.22 starpu_asynchronous_max_fpga_copy_disabled()

```
int starpu_asynchronous_max_fpga_copy_disabled (
            void )
```

Return 1 if asynchronous data transfers between CPU and Maxeler FPGA devices are disabled. See Maxeler FPGA Workers for more details.

57.15.4.23 starpu_asynchronous_mpi_ms_copy_disabled()

```
int starpu_asynchronous_mpi_ms_copy_disabled (
            void )
```

Return 1 if asynchronous data transfers between CPU and MPI Slave devices are disabled. See MPI Master Slave Workers for more details.

57.15.4.24 starpu_asynchronous_tcpip_ms_copy_disabled()

```
int starpu_asynchronous_tcpip_ms_copy_disabled (
            void )
```

Return 1 if asynchronous data transfers between CPU and TCP/IP Slave devices are disabled. See TCP/IP Master Slave Workers for more details.

57.15.4.25 starpu_asynchronous_copy_disabled_for()

```
int starpu_asynchronous_copy_disabled_for (
            enum starpu_node_kind kind )
```

Return 1 if asynchronous data transfers with a given kind of memory are disabled.

57.15.4.26 starpu_map_enabled()

```
int starpu_map_enabled (
            void )
```

Return 1 if memory mapping support between memory nodes is enabled. See General Configuration for more details.

57.15.4.27 starpu_display_stats()

```
void starpu_display_stats (
            void )
```

Call starpu_profiling_bus_helper_display_summary() and starpu_profiling_worker_helper_display_summary(). See Data Statistics for more details.

57.16 Task Insert Utility

Data Structures

- struct starpu_codelet_pack_arg_data

Macros

- #define **STARPU_MODE_SHIFT**
- #define STARPU_VALUE
- #define STARPU_CALLBACK
- #define STARPU_CALLBACK_WITH_ARG
- #define STARPU_CALLBACK_ARG
- #define STARPU_PRIORITY
- #define STARPU_DATA_ARRAY
- #define STARPU_DATA_MODE_ARRAY
- #define STARPU_TAG
- #define STARPU_HYPERVISOR_TAG
- #define STARPU_FLOPS
- #define STARPU_SCHED_CTX
- #define STARPU_PROLOGUE_CALLBACK
- #define STARPU_PROLOGUE_CALLBACK_ARG
- #define STARPU_PROLOGUE_CALLBACK_POP
- #define STARPU_PROLOGUE_CALLBACK_POP_ARG
- #define STARPU_EXECUTE_ON_WORKER
- #define STARPU_EXECUTE_WHERE
- #define STARPU_TAG_ONLY
- #define STARPU_POSSIBLY_PARALLEL
- #define STARPU_WORKER_ORDER
- #define STARPU_NAME
- #define STARPU_CL_ARGS
- #define STARPU_CL_ARGS_NFREE
- #define STARPU_TASK_DEPS_ARRAY
- #define STARPU_TASK_COLOR
- #define STARPU_HANDLES_SEQUENTIAL_CONSISTENCY
- #define STARPU_TASK_SYNCHRONOUS
- #define STARPU_TASK_END_DEPS_ARRAY
- #define STARPU_TASK_END_DEP
- #define STARPU_TASK_WORKERIDS
- #define STARPU_SEQUENTIAL_CONSISTENCY
- #define STARPU_TASK_PROFILING_INFO
- #define STARPU_TASK_NO_SUBMITORDER
- #define STARPU_CALLBACK_ARG_NFREE
- #define STARPU_CALLBACK_WITH_ARG_NFREE
- #define STARPU_PROLOGUE_CALLBACK_ARG_NFREE
- #define STARPU_PROLOGUE_CALLBACK_POP_ARG_NFREE
- #define STARPU_TASK_SCHED_DATA
- #define STARPU_TRANSACTION
- #define STARPU_TASK_FILE
- #define STARPU_TASK_LINE
- #define STARPU_EPILOGUE_CALLBACK
- #define STARPU_EPILOGUE_CALLBACK_ARG
- #define STARPU_SHIFTED_MODE_MAX

Functions

- int starpu_task_set (struct starpu_task ∗task, struct starpu_codelet ∗cl,...)
- struct starpu_task ∗ starpu_task_build (struct starpu_codelet ∗cl,...)
- int starpu_task_insert (struct starpu_codelet ∗cl,...)
- int starpu_insert_task (struct starpu_codelet ∗cl,...)
- void starpu_task_insert_data_make_room (struct starpu_codelet ∗cl, struct starpu_task ∗task, int ∗allocated_buffers, int current_buffer, int room)
- void starpu_task_insert_data_process_arg (struct starpu_codelet ∗cl, struct starpu_task ∗task, int ∗allocated_buffers, int ∗current_buffer, int arg_type, starpu_data_handle_t handle)
- void starpu_task_insert_data_process_array_arg (struct starpu_codelet ∗cl, struct starpu_task ∗task, int ∗allocated_buffers, int ∗current_buffer, int nb_handles, starpu_data_handle_t ∗handles)
- void starpu_task_insert_data_process_mode_array_arg (struct starpu_codelet ∗cl, struct starpu_task ∗task, int ∗allocated_buffers, int ∗current_buffer, int nb_descrs, struct starpu_data_descr ∗descrs)
- void starpu_codelet_pack_args (void ∗∗arg_buffer, size_t ∗arg_buffer_size,...)
- void starpu_codelet_pack_arg_init (struct starpu_codelet_pack_arg_data ∗state)
- void starpu_codelet_pack_arg (struct starpu_codelet_pack_arg_data ∗state, const void ∗ptr, size_t ptr_size)
- void starpu_codelet_pack_arg_fini (struct starpu_codelet_pack_arg_data ∗state, void ∗∗cl_arg, size_t ∗cl_arg_size)
- void starpu_codelet_unpack_args (void ∗cl_arg,...)
- void starpu_codelet_unpack_arg_init (struct starpu_codelet_pack_arg_data ∗state, void ∗cl_arg, size_t cl_arg_size)
- void starpu_codelet_unpack_arg (struct starpu_codelet_pack_arg_data ∗state, void ∗ptr, size_t size)
- void starpu_codelet_dup_arg (struct starpu_codelet_pack_arg_data ∗state, void ∗∗ptr, size_t ∗size)
- void starpu_codelet_pick_arg (struct starpu_codelet_pack_arg_data ∗state, void ∗∗ptr, size_t ∗size)
- void starpu_codelet_unpack_arg_fini (struct starpu_codelet_pack_arg_data ∗state)
- void starpu_codelet_unpack_discard_arg (struct starpu_codelet_pack_arg_data ∗state)
- void starpu_codelet_unpack_args_and_copyleft (void ∗cl_arg, void ∗buffer, size_t buffer_size,...)

57.16.1 Detailed Description

57.16.2 Data Structure Documentation

57.16.2.1 struct starpu_codelet_pack_arg_data

Structure to be used for starpu_codelet_pack_arg_init() & co, and starpu_codelet_unpack_arg_init() & co. The contents is public, however users should not directly access it, but only use as a parameter to the appropriate functions.

Data Fields

char ∗	arg_buffer	
size_t	arg_buffer_size	
size_t	arg_buffer_used	
size_t	current_offset	
int	nargs	

57.16.3 Macro Definition Documentation

57.16.3.1 STARPU_VALUE

`#define STARPU_VALUE`

Used when calling starpu_task_insert(), must be followed by a pointer to a constant value and the size of the constant

57.16.3.2 STARPU_CALLBACK

`#define STARPU_CALLBACK`
Used when calling starpu_task_insert(), must be followed by a pointer to a callback function

57.16.3.3 STARPU_CALLBACK_WITH_ARG

`#define STARPU_CALLBACK_WITH_ARG`
Used when calling starpu_task_insert(), must be followed by two pointers: one to a callback function, and the other to be given as an argument to the callback function; this is equivalent to using both STARPU_CALLBACK and STARPU_CALLBACK_ARG.

57.16.3.4 STARPU_CALLBACK_ARG

`#define STARPU_CALLBACK_ARG`
Used when calling starpu_task_insert(), must be followed by a pointer to be given as an argument to the callback function

57.16.3.5 STARPU_PRIORITY

`#define STARPU_PRIORITY`
Used when calling starpu_task_insert(), must be followed by a integer defining a priority level

57.16.3.6 STARPU_DATA_ARRAY

`#define STARPU_DATA_ARRAY`
Used when calling starpu_task_insert(), must be followed by an array of handles and the number of elements in the array (as int). This is equivalent to passing the handles as separate parameters with STARPU_R, STARPU_W or STARPU_RW.

57.16.3.7 STARPU_DATA_MODE_ARRAY

`#define STARPU_DATA_MODE_ARRAY`
Used when calling starpu_task_insert(), must be followed by an array of struct starpu_data_descr and the number of elements in the array (as int). This is equivalent to passing the handles with the corresponding modes.

57.16.3.8 STARPU_TAG

`#define STARPU_TAG`
Used when calling starpu_task_insert(), must be followed by a tag.

57.16.3.9 STARPU_HYPERVISOR_TAG

`#define STARPU_HYPERVISOR_TAG`
Used when calling starpu_task_insert(), must be followed by a tag.

57.16.3.10 STARPU_FLOPS

`#define STARPU_FLOPS`
Used when calling starpu_task_insert(), must be followed by an amount of floating point operations, as a double. Users **MUST** explicitly cast into double, otherwise parameter passing will not work.

57.16.3.11 STARPU_SCHED_CTX

`#define STARPU_SCHED_CTX`
Used when calling starpu_task_insert(), must be followed by the id of the scheduling context to which to submit the task to.

57.16.3.12 STARPU_PROLOGUE_CALLBACK

`#define STARPU_PROLOGUE_CALLBACK`

Used when calling starpu_task_insert(), must be followed by a pointer to a prologue callback function

57.16.3.13 STARPU_PROLOGUE_CALLBACK_ARG

`#define STARPU_PROLOGUE_CALLBACK_ARG`

Used when calling starpu_task_insert(), must be followed by a pointer to be given as an argument to the prologue callback function

57.16.3.14 STARPU_PROLOGUE_CALLBACK_POP

`#define STARPU_PROLOGUE_CALLBACK_POP`

Used when calling starpu_task_insert(), must be followed by a pointer to a prologue callback pop function

57.16.3.15 STARPU_PROLOGUE_CALLBACK_POP_ARG

`#define STARPU_PROLOGUE_CALLBACK_POP_ARG`

Used when calling starpu_task_insert(), must be followed by a pointer to be given as an argument to the prologue callback pop function

57.16.3.16 STARPU_EXECUTE_ON_WORKER

`#define STARPU_EXECUTE_ON_WORKER`

Used when calling starpu_task_insert(), must be followed by an integer value specifying the worker on which to execute the task (as specified by starpu_task::execute_on_a_specific_worker)

57.16.3.17 STARPU_EXECUTE_WHERE

`#define STARPU_EXECUTE_WHERE`

Used when calling starpu_task_insert(), must be followed by an unsigned long long value specifying the mask of worker on which to execute the task (as specified by starpu_task::where)

57.16.3.18 STARPU_TAG_ONLY

`#define STARPU_TAG_ONLY`

Used when calling starpu_task_insert(), must be followed by a tag stored in starpu_task::tag_id. Leave starpu_task::use_tag as 0.

57.16.3.19 STARPU_POSSIBLY_PARALLEL

`#define STARPU_POSSIBLY_PARALLEL`

Used when calling starpu_task_insert(), must be followed by an unsigned stored in starpu_task::possibly_parallel.

57.16.3.20 STARPU_WORKER_ORDER

`#define STARPU_WORKER_ORDER`

used when calling starpu_task_insert(), must be followed by an integer value specifying the worker order in which to execute the tasks (as specified by starpu_task::workerorder)

57.16.3.21 STARPU_NAME

`#define STARPU_NAME`

Used when calling starpu_task_insert(), must be followed by a char $*$ stored in starpu_task::name.

57.16.3.22 STARPU_CL_ARGS

`#define STARPU_CL_ARGS`

Used when calling starpu_task_insert(), must be followed by a memory buffer containing the arguments to be given to the task, and by the size of the arguments. The memory buffer should be the result of a previous call to starpu_codelet_pack_args(), and will be freed (i.e. starpu_task::cl_arg_free will be set to 1)

57.16.3.23 STARPU_CL_ARGS_NFREE

`#define STARPU_CL_ARGS_NFREE`

Used when calling starpu_task_insert(), similarly to STARPU_CL_ARGS, must be followed by a memory buffer containing the arguments to be given to the task, and by the size of the arguments. The memory buffer should be the result of a previous call to starpu_codelet_pack_args(), and will NOT be freed (i.e. starpu_task::cl_arg_free will be set to 0)

57.16.3.24 STARPU_TASK_DEPS_ARRAY

`#define STARPU_TASK_DEPS_ARRAY`

Used when calling starpu_task_insert(), must be followed by a number of tasks as int, and an array containing these tasks. The function starpu_task_declare_deps_array() will be called with the given values.

57.16.3.25 STARPU_TASK_COLOR

`#define STARPU_TASK_COLOR`

Used when calling starpu_task_insert(), must be followed by an integer representing a color

57.16.3.26 STARPU_HANDLES_SEQUENTIAL_CONSISTENCY

`#define STARPU_HANDLES_SEQUENTIAL_CONSISTENCY`

Used when calling starpu_task_insert(), must be followed by an array of characters representing the sequential consistency for each buffer of the task.

57.16.3.27 STARPU_TASK_SYNCHRONOUS

`#define STARPU_TASK_SYNCHRONOUS`

Used when calling starpu_task_insert(), must be followed by an integer stating if the task is synchronous or not

57.16.3.28 STARPU_TASK_END_DEPS_ARRAY

`#define STARPU_TASK_END_DEPS_ARRAY`

Used when calling starpu_task_insert(), must be followed by a number of tasks as int, and an array containing these tasks. The function starpu_task_declare_end_deps_array() will be called with the given values.

57.16.3.29 STARPU_TASK_END_DEP

`#define STARPU_TASK_END_DEP`

Used when calling starpu_task_insert(), must be followed by an integer which will be given to starpu_task_end_dep_add()

57.16.3.30 STARPU_TASK_WORKERIDS

`#define STARPU_TASK_WORKERIDS`

Used when calling starpu_task_insert(), must be followed by an unsigned being a number of workers, and an array of bits which size is the number of workers, the array indicates the set of workers which are allowed to execute the task.

57.16.3.31 STARPU_SEQUENTIAL_CONSISTENCY

`#define STARPU_SEQUENTIAL_CONSISTENCY`

Used when calling starpu_task_insert(), must be followed by an unsigned which sets the sequential consistency for the data parameters of the task.

57.16.3.32 STARPU_TASK_PROFILING_INFO

`#define STARPU_TASK_PROFILING_INFO`

Used when calling starpu_task_insert() and alike, must be followed by a pointer to a struct starpu_profiling_task_info

57.16.3.33 STARPU_TASK_NO_SUBMITORDER

`#define STARPU_TASK_NO_SUBMITORDER`

Used when calling starpu_task_insert() and alike, must be followed by an unsigned specifying not to allocate a submitorder id for the task

57.16.3.34 STARPU_CALLBACK_ARG_NFREE

`#define STARPU_CALLBACK_ARG_NFREE`

Used when calling starpu_task_insert(), similarly to STARPU_CALLBACK_ARG, must be followed by a pointer to be given as an argument to the callback function, the argument will not be freed, i.e starpu_task::callback_arg_free will be set to 0

57.16.3.35 STARPU_CALLBACK_WITH_ARG_NFREE

`#define STARPU_CALLBACK_WITH_ARG_NFREE`

Used when calling starpu_task_insert(), similarly to STARPU_CALLBACK_WITH_ARG, must be followed by two pointers: one to a callback function, and the other to be given as an argument to the callback function; this is equivalent to using both STARPU_CALLBACK and STARPU_CALLBACK_ARG_NFREE.

57.16.3.36 STARPU_PROLOGUE_CALLBACK_ARG_NFREE

`#define STARPU_PROLOGUE_CALLBACK_ARG_NFREE`

Used when calling starpu_task_insert(), similarly to STARPU_PROLOGUE_CALLBACK_ARG, must be followed by a pointer to be given as an argument to the prologue callback function, the argument will not be freed, i.e starpu_task::prologue_callback_arg_free will be set to 0

57.16.3.37 STARPU_PROLOGUE_CALLBACK_POP_ARG_NFREE

`#define STARPU_PROLOGUE_CALLBACK_POP_ARG_NFREE`

Used when calling starpu_task_insert(), similarly to STARPU_PROLOGUE_CALLBACK_POP_ARG, must be followed by a pointer to be given as an argument to the prologue callback pop function, the argument will not be freed, i.e starpu_task::prologue_callback_pop_arg_free will be set to 0

57.16.3.38 STARPU_TASK_SCHED_DATA

`#define STARPU_TASK_SCHED_DATA`

Used when calling starpu_task_insert() and alike, must be followed by a void∗ specifying the value to be set in starpu_task::sched_data

57.16.3.39 STARPU_TRANSACTION

`#define STARPU_TRANSACTION`

Used when calling starpu_task_insert() and alike, must be followed by a struct starpu_transaction ∗ specifying the value to be set in the transaction field of the task.

57.16.3.40 STARPU_TASK_FILE

`#define STARPU_TASK_FILE`

Used when calling starpu_task_insert(), must be followed by a char ∗ stored in starpu_task::file.
This is automatically set when FXT is enabled.

57.16.3.41 STARPU_TASK_LINE

`#define STARPU_TASK_LINE`

Used when calling starpu_task_insert(), must be followed by an int stored in starpu_task::line.
This is automatically set when FXT is enabled.

57.16.3.42 STARPU_EPILOGUE_CALLBACK

`#define STARPU_EPILOGUE_CALLBACK`

Used when calling starpu_task_insert(), must be followed by a pointer to a epilogue callback function

57.16.3.43 STARPU_EPILOGUE_CALLBACK_ARG

`#define STARPU_EPILOGUE_CALLBACK_ARG`

Used when calling starpu_task_insert(), must be followed by a pointer to be given as an argument to the epilogue callback function

57.16.3.44 STARPU_SHIFTED_MODE_MAX

`#define STARPU_SHIFTED_MODE_MAX`

This has to be the last mode value plus 1

57.16.4 Function Documentation

57.16.4.1 starpu_task_set()

```
int starpu_task_set (
            struct starpu_task * task,
            struct starpu_codelet * cl,
             ... )
```

Set the given `task` corresponding to `cl` with the following arguments. The argument list must be zero-terminated. The arguments following the codelet are the same as the ones for the function starpu_task_insert(). If some arguments of type STARPU_VALUE are given, the parameter starpu_task::cl_arg_free will be set to 1. See Other Task Utility Functions for more details.

57.16.4.2 starpu_task_build()

```
struct starpu_task * starpu_task_build (
            struct starpu_codelet * cl,
             ... )
```

Create a task corresponding to `cl` with the following arguments. The argument list must be zero-terminated. The arguments following the codelet are the same as the ones for the function starpu_task_insert(). If some arguments of type STARPU_VALUE are given, the parameter starpu_task::cl_arg_free will be set to 1. See Other Task Utility Functions for more details.

57.16.4.3 starpu_task_insert()

```
int starpu_task_insert (
            struct starpu_codelet * cl,
             ... )
```

Create and submit a task corresponding to `cl` with the following given arguments. The argument list must be zero-terminated.
The arguments following the codelet can be of the following types:

- STARPU_R, STARPU_W, STARPU_RW, STARPU_SCRATCH, STARPU_REDUX an access mode followed by a data handle;
- STARPU_DATA_ARRAY followed by an array of data handles and its number of elements;

- STARPU_DATA_MODE_ARRAY followed by an array of struct starpu_data_descr, i.e data handles with their associated access modes, and its number of elements;
- STARPU_EXECUTE_ON_WORKER, STARPU_WORKER_ORDER followed by an integer value specifying the worker on which to execute the task (as specified by starpu_task::execute_on_a_specific_worker)
- the specific values STARPU_VALUE, STARPU_CALLBACK, STARPU_CALLBACK_ARG, STARPU_CALLBACK_WITH_ARG, STARPU_PRIORITY, STARPU_TAG, STARPU_TAG_ONLY, STARPU_FLOPS, STARPU_SCHED_CTX, STARPU_CL_ARGS, STARPU_CL_ARGS_NFREE, STARPU_TASK_DEPS_ARRAY, STARPU_TASK_COLOR, STARPU_HANDLES_SEQUENTIAL_CONSISTENCY, STARPU_TASK_SYNCHRONOUS, STARPU_TASK_END_DEP followed by the appropriated objects as defined elsewhere.

When using STARPU_DATA_ARRAY, the access mode of the data handles is not defined, it will be taken from the codelet starpu_codelet::modes or starpu_codelet::dyn_modes field. One should use STARPU_DATA_MODE_ARRAY to define the data handles along with the access modes.
Parameters to be passed to the codelet implementation are defined through the type STARPU_VALUE. The function starpu_codelet_unpack_args() must be called within the codelet implementation to retrieve them.
See Insert Task Utility for more details.

57.16.4.4 starpu_insert_task()

```
int starpu_insert_task (
             struct starpu_codelet * cl,
              ... )
```

Identical to starpu_task_insert(). Kept to avoid breaking old codes.

57.16.4.5 starpu_task_insert_data_make_room()

```
void starpu_task_insert_data_make_room (
             struct starpu_codelet * cl,
             struct starpu_task * task,
             int * allocated_buffers,
             int current_buffer,
             int room )
```

Assuming that there are already `current_buffer` data handles passed to the task, and if ∗allocated_buffers is not 0, the `task->dyn_handles` array has size `∗allocated_buffers`, this function makes room for `room` other data handles, allocating or reallocating `task->dyn_handles` as necessary and updating `allocated_buffers` accordingly. One can thus start with allocated_buffers equal to 0 and current_buffer equal to 0, then make room by calling this function, then store handles with STARPU_TASK_SET_HANDLE(), make room again with this function, store yet more handles, etc. See Other Task Utility Functions for more details.

57.16.4.6 starpu_task_insert_data_process_arg()

```
void starpu_task_insert_data_process_arg (
             struct starpu_codelet * cl,
             struct starpu_task * task,
             int * allocated_buffers,
             int * current_buffer,
             int arg_type,
             starpu_data_handle_t handle )
```

Store data handle `handle` into task `task` with mode `arg_type`, updating `∗allocated_buffers` and `∗current_buffer` accordingly. See Other Task Utility Functions for more details.

57.16.4.7 starpu_task_insert_data_process_array_arg()

```
void starpu_task_insert_data_process_array_arg (
             struct starpu_codelet * cl,
             struct starpu_task * task,
             int * allocated_buffers,
             int * current_buffer,
             int nb_handles,
             starpu_data_handle_t * handles )
```

Store `nb_handles` data handles `handles` into task `task`, updating `*allocated_buffers` and `*current_buffer` accordingly. See Other Task Utility Functions for more details.

57.16.4.8 starpu_task_insert_data_process_mode_array_arg()

```
void starpu_task_insert_data_process_mode_array_arg (
            struct starpu_codelet * cl,
            struct starpu_task * task,
            int * allocated_buffers,
            int * current_buffer,
            int nb_descrs,
            struct starpu_data_descr * descrs )
```

Store `nb_descrs` data handles described by `descrs` into task `task`, updating `*allocated_buffers` and `*current_buffer` accordingly. See Other Task Utility Functions for more details.

57.16.4.9 starpu_codelet_pack_args()

```
void starpu_codelet_pack_args (
            void ** arg_buffer,
            size_t * arg_buffer_size,
             ... )
```

Pack arguments of type STARPU_VALUE into a buffer which can be given to a codelet and later unpacked with the function starpu_codelet_unpack_args().

Instead of calling starpu_codelet_pack_args(), one can also call starpu_codelet_pack_arg_init(), then starpu_codelet_pack_arg() for each data, then starpu_codelet_pack_arg_fini().

See Insert Task Utility for more details.

57.16.4.10 starpu_codelet_pack_arg_init()

```
void starpu_codelet_pack_arg_init (
            struct starpu_codelet_pack_arg_data * state )
```

Initialize struct starpu_codelet_pack_arg before calling starpu_codelet_pack_arg() and starpu_codelet_pack_arg_fini(). This will simply initialize the content of the structure. See Insert Task Utility for more details.

57.16.4.11 starpu_codelet_pack_arg()

```
void starpu_codelet_pack_arg (
            struct starpu_codelet_pack_arg_data * state,
            const void * ptr,
            size_t ptr_size )
```

Pack one argument into struct starpu_codelet_pack_arg `state`. That structure has to be initialized before with starpu_codelet_pack_arg_init(), and after all starpu_codelet_pack_arg() calls performed, starpu_codelet_pack_arg_fini() has to be used to get the `cl_arg` and `cl_arg_size` to be put in the task. See Insert Task Utility for more details.

57.16.4.12 starpu_codelet_pack_arg_fini()

```
void starpu_codelet_pack_arg_fini (
            struct starpu_codelet_pack_arg_data * state,
            void ** cl_arg,
            size_t * cl_arg_size )
```

Finish packing data, after calling starpu_codelet_pack_arg_init() once and starpu_codelet_pack_arg() several times. See Insert Task Utility for more details.

57.16.4.13 starpu_codelet_unpack_args()

```
void starpu_codelet_unpack_args (
            void * cl_arg,
             ... )
```

Retrieve the arguments of type STARPU_VALUE associated to a task automatically created using the function starpu_task_insert(). If any parameter's value is 0, unpacking will stop there and ignore the remaining parameters. See Insert Task Utility for more details.

57.16.4.14 starpu_codelet_unpack_arg_init()

```
void starpu_codelet_unpack_arg_init (
            struct starpu_codelet_pack_arg_data * state,
            void * cl_arg,
            size_t cl_arg_size )
```

Initialize `state` with `cl_arg` and `cl_arg_size`. This has to be called before calling starpu_codelet_unpack_arg(). See Insert Task Utility for more details.

57.16.4.15 starpu_codelet_unpack_arg()

```
void starpu_codelet_unpack_arg (
            struct starpu_codelet_pack_arg_data * state,
            void * ptr,
            size_t size )
```

Unpack the next argument of size `size` from `state` into `ptr` with a copy. `state` has to be initialized before with starpu_codelet_unpack_arg_init(). See Insert Task Utility for more details.

57.16.4.16 starpu_codelet_dup_arg()

```
void starpu_codelet_dup_arg (
            struct starpu_codelet_pack_arg_data * state,
            void ** ptr,
            size_t * size )
```

Unpack the next argument of unknown size from `state` into `ptr` with a copy. `ptr` is allocated before copying in it the value of the argument. The size of the argument is returned in `size`. `has` to be initialized before with starpu_codelet_unpack_arg_init(). See Insert Task Utility for more details.

57.16.4.17 starpu_codelet_pick_arg()

```
void starpu_codelet_pick_arg (
            struct starpu_codelet_pack_arg_data * state,
            void ** ptr,
            size_t * size )
```

Unpack the next argument of unknown size from `state` into `ptr`. `ptr` will be a pointer to the memory of the argument. The size of the argument is returned in `size`. `has` to be initialized before with starpu_codelet_unpack_arg_init(). See Insert Task Utility for more details.

57.16.4.18 starpu_codelet_unpack_arg_fini()

```
void starpu_codelet_unpack_arg_fini (
            struct starpu_codelet_pack_arg_data * state )
```

Finish unpacking data, after calling starpu_codelet_unpack_arg_init() once and starpu_codelet_unpack_arg() or starpu_codelet_dup_arg() or starpu_codelet_pick_arg() several times. See Insert Task Utility for more details.

57.16.4.19 starpu_codelet_unpack_discard_arg()

```
void starpu_codelet_unpack_discard_arg (
            struct starpu_codelet_pack_arg_data * state )
```

Call this function during unpacking to skip saving the argument in ptr. See Insert Task Utility for more details.

57.16.4.20 starpu_codelet_unpack_args_and_copyleft()

```
void starpu_codelet_unpack_args_and_copyleft (
            void * cl_arg,
            void * buffer,
            size_t buffer_size,
             ... )
```

Similar to starpu_codelet_unpack_args(), but if any parameter is 0, copy the part of `cl_arg` that has not been read in `buffer` which can then be used in a later call to one of the unpack functions. See Insert Task Utility for more details.

57.17 Interoperability Support

API to interoperate with other runtime systems.

Typedefs

- typedef int **starpurm_drs_ret_t**
- typedef void ∗ **starpurm_drs_desc_t**
- typedef void ∗ **starpurm_drs_cbs_t**
- typedef void(∗ **starpurm_drs_cb_t**) (void ∗)
- typedef void ∗ **starpurm_block_cond_t**
- typedef int(∗ **starpurm_polling_t**) (void ∗)

Enumerations

- enum e_starpurm_drs_ret { starpurm_DRS_SUCCESS , starpurm_DRS_DISABLD , starpurm_DRS_PERM , starpurm_DRS_EINVAL }

Initialisation

- void starpurm_initialize_with_cpuset (hwloc_cpuset_t initially_owned_cpuset)
- void starpurm_initialize (void)
- void starpurm_shutdown (void)

Spawn

- void starpurm_spawn_kernel_on_cpus (void ∗data, void(∗f)(void ∗), void ∗args, hwloc_cpuset_t cpuset)
- void starpurm_spawn_kernel_on_cpus_callback (void ∗data, void(∗f)(void ∗), void ∗args, hwloc_cpuset_t cpuset, void(∗cb_f)(void ∗), void ∗cb_args)
- void **starpurm_spawn_kernel_callback** (void ∗data, void(∗f)(void ∗), void ∗args, void(∗cb_f)(void ∗), void ∗cb_args)

DynamicResourceSharing

- starpurm_drs_ret_t starpurm_set_drs_enable (starpurm_drs_desc_t ∗spd)
- starpurm_drs_ret_t starpurm_set_drs_disable (starpurm_drs_desc_t ∗spd)
- int starpurm_drs_enabled_p (void)
- starpurm_drs_ret_t starpurm_set_max_parallelism (starpurm_drs_desc_t ∗spd, int max)
- starpurm_drs_ret_t starpurm_assign_cpu_to_starpu (starpurm_drs_desc_t ∗spd, int cpuid)
- starpurm_drs_ret_t starpurm_assign_cpus_to_starpu (starpurm_drs_desc_t ∗spd, int ncpus)
- starpurm_drs_ret_t starpurm_assign_cpu_mask_to_starpu (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_t mask)
- starpurm_drs_ret_t starpurm_assign_all_cpus_to_starpu (starpurm_drs_desc_t ∗spd)
- starpurm_drs_ret_t starpurm_withdraw_cpu_from_starpu (starpurm_drs_desc_t ∗spd, int cpuid)
- starpurm_drs_ret_t starpurm_withdraw_cpus_from_starpu (starpurm_drs_desc_t ∗spd, int ncpus)
- starpurm_drs_ret_t starpurm_withdraw_cpu_mask_from_starpu (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_t mask)
- starpurm_drs_ret_t starpurm_withdraw_all_cpus_from_starpu (starpurm_drs_desc_t ∗spd)
- starpurm_drs_ret_t starpurm_lend (starpurm_drs_desc_t ∗spd)
- starpurm_drs_ret_t starpurm_lend_cpu (starpurm_drs_desc_t ∗spd, int cpuid)
- starpurm_drs_ret_t starpurm_lend_cpus (starpurm_drs_desc_t ∗spd, int ncpus)
- starpurm_drs_ret_t starpurm_lend_cpu_mask (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_t mask)
- starpurm_drs_ret_t starpurm_reclaim (starpurm_drs_desc_t ∗spd)
- starpurm_drs_ret_t starpurm_reclaim_cpu (starpurm_drs_desc_t ∗spd, int cpuid)
- starpurm_drs_ret_t starpurm_reclaim_cpus (starpurm_drs_desc_t ∗spd, int ncpus)
- starpurm_drs_ret_t starpurm_reclaim_cpu_mask (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_t mask)

- starpurm_drs_ret_t starpurm_acquire (starpurm_drs_desc_t ∗spd)
- starpurm_drs_ret_t starpurm_acquire_cpu (starpurm_drs_desc_t ∗spd, int cpuid)
- starpurm_drs_ret_t starpurm_acquire_cpus (starpurm_drs_desc_t ∗spd, int ncpus)
- starpurm_drs_ret_t starpurm_acquire_cpu_mask (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_t mask)
- starpurm_drs_ret_t starpurm_return_all (starpurm_drs_desc_t ∗spd)
- starpurm_drs_ret_t starpurm_return_cpu (starpurm_drs_desc_t ∗spd, int cpuid)

Devices

- int starpurm_get_device_type_id (const char ∗type_str)
- const char ∗ starpurm_get_device_type_name (int type_id)
- int starpurm_get_nb_devices_by_type (int type_id)
- int starpurm_get_device_id (int type_id, int device_rank)
- starpurm_drs_ret_t starpurm_assign_device_to_starpu (starpurm_drs_desc_t ∗spd, int type_id, int unit_↩ rank)
- starpurm_drs_ret_t starpurm_assign_devices_to_starpu (starpurm_drs_desc_t ∗spd, int type_id, int ndevices)
- starpurm_drs_ret_t starpurm_assign_device_mask_to_starpu (starpurm_drs_desc_t ∗spd, const hwloc_↩ cpuset_t mask)
- starpurm_drs_ret_t starpurm_assign_all_devices_to_starpu (starpurm_drs_desc_t ∗spd, int type_id)
- starpurm_drs_ret_t starpurm_withdraw_device_from_starpu (starpurm_drs_desc_t ∗spd, int type_id, int unit_rank)
- starpurm_drs_ret_t starpurm_withdraw_devices_from_starpu (starpurm_drs_desc_t ∗spd, int type_id, int ndevices)
- starpurm_drs_ret_t starpurm_withdraw_device_mask_from_starpu (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_t mask)
- starpurm_drs_ret_t starpurm_withdraw_all_devices_from_starpu (starpurm_drs_desc_t ∗spd, int type_id)
- starpurm_drs_ret_t starpurm_lend_device (starpurm_drs_desc_t ∗spd, int type_id, int unit_rank)
- starpurm_drs_ret_t starpurm_lend_devices (starpurm_drs_desc_t ∗spd, int type_id, int ndevices)
- starpurm_drs_ret_t starpurm_lend_device_mask (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_t mask)
- starpurm_drs_ret_t starpurm_lend_all_devices (starpurm_drs_desc_t ∗spd, int type_id)
- starpurm_drs_ret_t starpurm_reclaim_device (starpurm_drs_desc_t ∗spd, int type_id, int unit_rank)
- starpurm_drs_ret_t starpurm_reclaim_devices (starpurm_drs_desc_t ∗spd, int type_id, int ndevices)
- starpurm_drs_ret_t starpurm_reclaim_device_mask (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_↩ t mask)
- starpurm_drs_ret_t starpurm_reclaim_all_devices (starpurm_drs_desc_t ∗spd, int type_id)
- starpurm_drs_ret_t starpurm_acquire_device (starpurm_drs_desc_t ∗spd, int type_id, int unit_rank)
- starpurm_drs_ret_t starpurm_acquire_devices (starpurm_drs_desc_t ∗spd, int type_id, int ndevices)
- starpurm_drs_ret_t starpurm_acquire_device_mask (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_↩ t mask)
- starpurm_drs_ret_t starpurm_acquire_all_devices (starpurm_drs_desc_t ∗spd, int type_id)
- starpurm_drs_ret_t starpurm_return_all_devices (starpurm_drs_desc_t ∗spd, int type_id)
- starpurm_drs_ret_t starpurm_return_device (starpurm_drs_desc_t ∗spd, int type_id, int unit_rank)

CpusetsQueries

- hwloc_cpuset_t starpurm_get_device_worker_cpuset (int type_id, int unit_rank)
- hwloc_cpuset_t starpurm_get_global_cpuset (void)
- hwloc_cpuset_t starpurm_get_selected_cpuset (void)
- hwloc_cpuset_t starpurm_get_all_cpu_workers_cpuset (void)
- hwloc_cpuset_t starpurm_get_all_device_workers_cpuset (void)
- hwloc_cpuset_t starpurm_get_all_device_workers_cpuset_by_type (int typeid)

57.17.1 Detailed Description

API to interoperate with other runtime systems.

57.17.2 Enumeration Type Documentation

57.17.2.1 e_starpurm_drs_ret

`enum e_starpurm_drs_ret`

StarPU Resource Manager return type.

Enumerator

starpurm_DRS_SUCCESS	Dynamic resource sharing operation succeeded.
starpurm_DRS_DISABLD	Dynamic resource sharing is disabled.
starpurm_DRS_PERM	Dynamic resource sharing operation is not authorized or implemented.
starpurm_DRS_EINVAL	Dynamic resource sharing operation has been called with one or more invalid parameters.

57.17.3 Function Documentation

57.17.3.1 starpurm_initialize_with_cpuset()

```
void starpurm_initialize_with_cpuset (
            hwloc_cpuset_t initially_owned_cpuset )
```

Resource enforcement

57.17.3.2 starpurm_initialize()

```
void starpurm_initialize (
            void  )
```

Initialize StarPU and the StarPU-RM resource management module. The starpu_init() function should not have been called before the call to starpurm_initialize(). The starpurm_initialize() function will take care of this

57.17.3.3 starpurm_shutdown()

```
void starpurm_shutdown (
            void  )
```

Shutdown StarPU-RM and StarPU. The starpu_shutdown() function should not be called before. The starpurm_shutdown() function will take care of this.

57.17.3.4 starpurm_spawn_kernel_on_cpus()

```
void starpurm_spawn_kernel_on_cpus (
            void * data,
            void(*)(void *) f,
            void * args,
            hwloc_cpuset_t cpuset )
```

Allocate a temporary context spanning the units selected in the cpuset bitmap, set it as the default context for the current thread, and call user function `f`. Upon the return of user function `f`, the temporary context is freed and the previous default context for the current thread is restored.

57.17.3.5 starpurm_spawn_kernel_on_cpus_callback()

```
void starpurm_spawn_kernel_on_cpus_callback (
            void * data,
            void(*)(void *) f,
            void * args,
            hwloc_cpuset_t cpuset,
            void(*)(void *) cb_f,
            void * cb_args )
```

Spawn a POSIX thread and returns immediately. The thread spawned will allocate a temporary context spanning the units selected in the cpuset bitmap, set it as the default context for the current thread, and call user function `f`. Upon the return of user function `f`, the temporary context will be freed and the previous default context for the current thread restored. A user specified callback `cb_f` will be called just before the termination of the thread.

57.17.3.6 starpurm_set_drs_enable()

```
starpurm_drs_ret_t starpurm_set_drs_enable (
            starpurm_drs_desc_t * spd )
```

Turn-on dynamic resource sharing support.

57.17.3.7 starpurm_set_drs_disable()

```
starpurm_drs_ret_t starpurm_set_drs_disable (
            starpurm_drs_desc_t * spd )
```

Turn-off dynamic resource sharing support.

57.17.3.8 starpurm_drs_enabled_p()

```
int starpurm_drs_enabled_p (
            void )
```

Return the state of the dynamic resource sharing support (`=!0` enabled, `=0` disabled).

57.17.3.9 starpurm_set_max_parallelism()

```
starpurm_drs_ret_t starpurm_set_max_parallelism (
            starpurm_drs_desc_t * spd,
            int max )
```

Set the maximum number of CPU computing units available for StarPU computations to `max`. This number cannot exceed the maximum number of StarPU's CPU worker allocated at start-up time.

57.17.3.10 starpurm_assign_cpu_to_starpu()

```
starpurm_drs_ret_t starpurm_assign_cpu_to_starpu (
            starpurm_drs_desc_t * spd,
            int cpuid )
```

Extend StarPU's default scheduling context to execute tasks on worker corresponding to logical unit `cpuid`. If StarPU does not have a worker thread initialized for logical unit `cpuid`, do nothing.

57.17.3.11 starpurm_assign_cpus_to_starpu()

```
starpurm_drs_ret_t starpurm_assign_cpus_to_starpu (
            starpurm_drs_desc_t * spd,
            int ncpus )
```

Extend StarPU's default scheduling context to execute tasks on `ncpus` more workers, up to the number of StarPU worker threads initialized.

57.17.3.12 starpurm_assign_cpu_mask_to_starpu()

```
starpurm_drs_ret_t starpurm_assign_cpu_mask_to_starpu (
            starpurm_drs_desc_t * spd,
            const hwloc_cpuset_t mask )
```

Extend StarPU's default scheduling context to execute tasks on the additional logical units selected in `mask`. Logical units of `mask` for which no StarPU worker is initialized are silently ignored.

57.17.3.13 starpurm_assign_all_cpus_to_starpu()

```
starpurm_drs_ret_t starpurm_assign_all_cpus_to_starpu (
            starpurm_drs_desc_t * spd )
```

Set StarPU's default scheduling context to execute tasks on all available logical units for which a StarPU worker has been initialized.

57.17.3.14 starpurm_withdraw_cpu_from_starpu()

```
starpurm_drs_ret_t starpurm_withdraw_cpu_from_starpu (
            starpurm_drs_desc_t * spd,
            int cpuid )
```

Shrink StarPU's default scheduling context so as to not execute tasks on worker corresponding to logical unit `cpuid`. If StarPU does not have a worker thread initialized for logical unit `cpuid`, do nothing.

57.17.3.15 starpurm_withdraw_cpus_from_starpu()

```
starpurm_drs_ret_t starpurm_withdraw_cpus_from_starpu (
            starpurm_drs_desc_t * spd,
            int ncpus )
```

Shrink StarPU's default scheduling context to execute tasks on `ncpus` less workers.

57.17.3.16 starpurm_withdraw_cpu_mask_from_starpu()

```
starpurm_drs_ret_t starpurm_withdraw_cpu_mask_from_starpu (
            starpurm_drs_desc_t * spd,
            const hwloc_cpuset_t mask )
```

Shrink StarPU's default scheduling context so as to not execute tasks on the logical units selected in `mask`. Logical units of `mask` for which no StarPU worker is initialized are silently ignored.

57.17.3.17 starpurm_withdraw_all_cpus_from_starpu()

```
starpurm_drs_ret_t starpurm_withdraw_all_cpus_from_starpu (
            starpurm_drs_desc_t * spd )
```

Shrink StarPU's default scheduling context so as to remove all logical units.

57.17.3.18 starpurm_lend()

```
starpurm_drs_ret_t starpurm_lend (
            starpurm_drs_desc_t * spd )
```

Synonym for starpurm_assign_all_cpus_to_starpu().

57.17.3.19 starpurm_lend_cpu()

```
starpurm_drs_ret_t starpurm_lend_cpu (
            starpurm_drs_desc_t * spd,
            int cpuid )
```

Synonym for starpurm_assign_cpu_to_starpu().

57.17.3.20 starpurm_lend_cpus()

```
starpurm_drs_ret_t starpurm_lend_cpus (
            starpurm_drs_desc_t * spd,
            int ncpus )
```

Synonym for starpurm_assign_cpus_to_starpu().

57.17.3.21 starpurm_lend_cpu_mask()

```
starpurm_drs_ret_t starpurm_lend_cpu_mask (
             starpurm_drs_desc_t * spd,
             const hwloc_cpuset_t mask )
```

Synonym for starpurm_assign_cpu_mask_to_starpu().

57.17.3.22 starpurm_reclaim()

```
starpurm_drs_ret_t starpurm_reclaim (
             starpurm_drs_desc_t * spd )
```

Synonym for starpurm_withdraw_all_cpus_from_starpu().

57.17.3.23 starpurm_reclaim_cpu()

```
starpurm_drs_ret_t starpurm_reclaim_cpu (
             starpurm_drs_desc_t * spd,
             int cpuid )
```

Synonym for starpurm_withdraw_cpu_from_starpu().

57.17.3.24 starpurm_reclaim_cpus()

```
starpurm_drs_ret_t starpurm_reclaim_cpus (
             starpurm_drs_desc_t * spd,
             int ncpus )
```

Synonym for starpurm_withdraw_cpus_from_starpu().

57.17.3.25 starpurm_reclaim_cpu_mask()

```
starpurm_drs_ret_t starpurm_reclaim_cpu_mask (
             starpurm_drs_desc_t * spd,
             const hwloc_cpuset_t mask )
```

Synonym for starpurm_withdraw_cpu_mask_from_starpu().

57.17.3.26 starpurm_acquire()

```
starpurm_drs_ret_t starpurm_acquire (
             starpurm_drs_desc_t * spd )
```

Synonym for starpurm_withdraw_all_cpus_from_starpu().

57.17.3.27 starpurm_acquire_cpu()

```
starpurm_drs_ret_t starpurm_acquire_cpu (
             starpurm_drs_desc_t * spd,
             int cpuid )
```

Synonym for starpurm_withdraw_cpu_from_starpu().

57.17.3.28 starpurm_acquire_cpus()

```
starpurm_drs_ret_t starpurm_acquire_cpus (
             starpurm_drs_desc_t * spd,
             int ncpus )
```

Synonym for starpurm_withdraw_cpus_from_starpu().

57.17.3.29 starpurm_acquire_cpu_mask()

```
starpurm_drs_ret_t starpurm_acquire_cpu_mask (
             starpurm_drs_desc_t * spd,
             const hwloc_cpuset_t mask )
```

Synonym for starpurm_withdraw_cpu_mask_from_starpu().

57.17.3.30 starpurm_return_all()

```
starpurm_drs_ret_t starpurm_return_all (
              starpurm_drs_desc_t * spd )
```

Synonym for starpurm_assign_all_cpus_to_starpu().

57.17.3.31 starpurm_return_cpu()

```
starpurm_drs_ret_t starpurm_return_cpu (
              starpurm_drs_desc_t * spd,
              int cpuid )
```

Synonym for starpurm_assign_cpu_to_starpu().

57.17.3.32 starpurm_get_device_type_id()

```
int starpurm_get_device_type_id (
              const char * type_str )
```

Return the device type ID constant associated to the device type name. Valid names for `type_str` are:

- `"cpu"`: regular CPU unit;
- `"opencl"`: OpenCL device unit;
- `"cuda"`: nVidia CUDA device unit;

57.17.3.33 starpurm_get_device_type_name()

```
const char * starpurm_get_device_type_name (
              int type_id )
```

Return the device type name associated to the device type ID constant.

57.17.3.34 starpurm_get_nb_devices_by_type()

```
int starpurm_get_nb_devices_by_type (
              int type_id )
```

Return the number of initialized StarPU worker for the device type `type_id`.

57.17.3.35 starpurm_get_device_id()

```
int starpurm_get_device_id (
              int type_id,
              int device_rank )
```

Return the unique ID assigned to the `device_rank` nth device of type `type_id`.

57.17.3.36 starpurm_assign_device_to_starpu()

```
starpurm_drs_ret_t starpurm_assign_device_to_starpu (
              starpurm_drs_desc_t * spd,
              int type_id,
              int unit_rank )
```

Extend StarPU's default scheduling context to use `unit_rank` nth device of type `type_id`.

57.17.3.37 starpurm_assign_devices_to_starpu()

```
starpurm_drs_ret_t starpurm_assign_devices_to_starpu (
              starpurm_drs_desc_t * spd,
              int type_id,
              int ndevices )
```

Extend StarPU's default scheduling context to use `ndevices` more devices of type `type_id`, up to the number of StarPU workers initialized for such device type.

57.17.3.38 starpurm_assign_device_mask_to_starpu()

```
starpurm_drs_ret_t starpurm_assign_device_mask_to_starpu (
            starpurm_drs_desc_t * spd,
            const hwloc_cpuset_t mask )
```

Extend StarPU's default scheduling context to use additional devices as designated by their corresponding StarPU worker thread(s) CPU-set `mask`.

57.17.3.39 starpurm_assign_all_devices_to_starpu()

```
starpurm_drs_ret_t starpurm_assign_all_devices_to_starpu (
            starpurm_drs_desc_t * spd,
            int type_id )
```

Extend StarPU's default scheduling context to use all devices of type `type_id` for which it has a worker thread initialized.

57.17.3.40 starpurm_withdraw_device_from_starpu()

```
starpurm_drs_ret_t starpurm_withdraw_device_from_starpu (
            starpurm_drs_desc_t * spd,
            int type_id,
            int unit_rank )
```

Shrink StarPU's default scheduling context to not use `unit_rank` nth device of type `type_id`.

57.17.3.41 starpurm_withdraw_devices_from_starpu()

```
starpurm_drs_ret_t starpurm_withdraw_devices_from_starpu (
            starpurm_drs_desc_t * spd,
            int type_id,
            int ndevices )
```

Shrink StarPU's default scheduling context to use `ndevices` less devices of type `type_id`.

57.17.3.42 starpurm_withdraw_device_mask_from_starpu()

```
starpurm_drs_ret_t starpurm_withdraw_device_mask_from_starpu (
            starpurm_drs_desc_t * spd,
            const hwloc_cpuset_t mask )
```

Shrink StarPU's default scheduling context to not use devices designated by their corresponding StarPU worker thread(s) CPU-set `mask`.

57.17.3.43 starpurm_withdraw_all_devices_from_starpu()

```
starpurm_drs_ret_t starpurm_withdraw_all_devices_from_starpu (
            starpurm_drs_desc_t * spd,
            int type_id )
```

Shrink StarPU's default scheduling context to use no devices of type `type_id`.

57.17.3.44 starpurm_lend_device()

```
starpurm_drs_ret_t starpurm_lend_device (
            starpurm_drs_desc_t * spd,
            int type_id,
            int unit_rank )
```

Synonym for starpurm_assign_device_to_starpu().

57.17.3.45 starpurm_lend_devices()

```
starpurm_drs_ret_t starpurm_lend_devices (
            starpurm_drs_desc_t * spd,
            int type_id,
            int ndevices )
```

Synonym for starpurm_assign_devices_to_starpu().

57.17.3.46 starpurm_lend_device_mask()

```
starpurm_drs_ret_t starpurm_lend_device_mask (
              starpurm_drs_desc_t * spd,
              const hwloc_cpuset_t mask )
```

Synonym for starpurm_assign_device_mask_to_starpu().

57.17.3.47 starpurm_lend_all_devices()

```
starpurm_drs_ret_t starpurm_lend_all_devices (
              starpurm_drs_desc_t * spd,
              int type_id )
```

Synonym for starpurm_assign_all_devices_to_starpu().

57.17.3.48 starpurm_reclaim_device()

```
starpurm_drs_ret_t starpurm_reclaim_device (
              starpurm_drs_desc_t * spd,
              int type_id,
              int unit_rank )
```

Synonym for starpurm_withdraw_device_from_starpu().

57.17.3.49 starpurm_reclaim_devices()

```
starpurm_drs_ret_t starpurm_reclaim_devices (
              starpurm_drs_desc_t * spd,
              int type_id,
              int ndevices )
```

Synonym for starpurm_withdraw_devices_from_starpu().

57.17.3.50 starpurm_reclaim_device_mask()

```
starpurm_drs_ret_t starpurm_reclaim_device_mask (
              starpurm_drs_desc_t * spd,
              const hwloc_cpuset_t mask )
```

Synonym for starpurm_withdraw_device_mask_from_starpu().

57.17.3.51 starpurm_reclaim_all_devices()

```
starpurm_drs_ret_t starpurm_reclaim_all_devices (
              starpurm_drs_desc_t * spd,
              int type_id )
```

Synonym for starpurm_withdraw_all_devices_from_starpu().

57.17.3.52 starpurm_acquire_device()

```
starpurm_drs_ret_t starpurm_acquire_device (
              starpurm_drs_desc_t * spd,
              int type_id,
              int unit_rank )
```

Synonym for starpurm_withdraw_device_from_starpu().

57.17.3.53 starpurm_acquire_devices()

```
starpurm_drs_ret_t starpurm_acquire_devices (
             starpurm_drs_desc_t * spd,
             int type_id,
             int ndevices )
```

Synonym for starpurm_withdraw_devices_from_starpu().

57.17.3.54 starpurm_acquire_device_mask()

```
starpurm_drs_ret_t starpurm_acquire_device_mask (
             starpurm_drs_desc_t * spd,
             const hwloc_cpuset_t mask )
```

Synonym for starpurm_withdraw_device_mask_from_starpu().

57.17.3.55 starpurm_acquire_all_devices()

```
starpurm_drs_ret_t starpurm_acquire_all_devices (
             starpurm_drs_desc_t * spd,
             int type_id )
```

Synonym for starpurm_withdraw_all_devices_from_starpu().

57.17.3.56 starpurm_return_all_devices()

```
starpurm_drs_ret_t starpurm_return_all_devices (
             starpurm_drs_desc_t * spd,
             int type_id )
```

Synonym for starpurm_assign_all_devices_to_starpu().

57.17.3.57 starpurm_return_device()

```
starpurm_drs_ret_t starpurm_return_device (
             starpurm_drs_desc_t * spd,
             int type_id,
             int unit_rank )
```

Synonym for starpurm_assign_device_to_starpu().

57.17.3.58 starpurm_get_device_worker_cpuset()

```
hwloc_cpuset_t starpurm_get_device_worker_cpuset (
             int type_id,
             int unit_rank )
```

Return the CPU-set of the StarPU worker associated to the `unit_rank` nth unit of type `type_id`.

57.17.3.59 starpurm_get_global_cpuset()

```
hwloc_cpuset_t starpurm_get_global_cpuset (
             void )
```

Return the cumulated CPU-set of all StarPU worker threads.

57.17.3.60 starpurm_get_selected_cpuset()

```
hwloc_cpuset_t starpurm_get_selected_cpuset (
             void )
```

Return the CPU-set of the StarPU worker threads currently selected in the default StarPU's scheduling context.

57.17.3.61 starpurm_get_all_cpu_workers_cpuset()

```
hwloc_cpuset_t starpurm_get_all_cpu_workers_cpuset (
             void )
```

Return the cumulated CPU-set of all CPU StarPU worker threads.

57.17.3.62 starpurm_get_all_device_workers_cpuset()

```
hwloc_cpuset_t starpurm_get_all_device_workers_cpuset (
            void  )
```

Return the cumulated CPU-set of all "non-CPU" StarPU worker threads.

57.17.3.63 starpurm_get_all_device_workers_cpuset_by_type()

```
hwloc_cpuset_t starpurm_get_all_device_workers_cpuset_by_type (
            int typeid )
```

Return the cumulated CPU-set of all StarPU worker threads for devices of type `typeid`.

57.18 Maxeler FPGA Extensions

Data Structures

- struct starpu_max_load

Macros

- #define STARPU_USE_MAX_FPGA
- #define STARPU_MAXMAXFPGADEVS

Functions

- max_engine_t ∗ starpu_max_fpga_get_local_engine (void)

57.18.1 Detailed Description

57.18.2 Data Structure Documentation

57.18.2.1 struct starpu_max_load

This specifies a Maxeler file to be loaded on some engines.

Data Fields

max_file_t ∗	file	Provide the file to be loaded
const char ∗	engine_id_pattern	Provide the engine(s) on which to be loaded, following the Maxeler engine naming, i.e. typically "∗:0", "∗:1", etc. In an array of struct starpu_max_load, only one can have the "∗" specification.

57.18.3 Macro Definition Documentation

57.18.3.1 STARPU_USE_MAX_FPGA

```
#define STARPU_USE_MAX_FPGA
```

Defined when StarPU has been installed with FPGA support. It should be used in your code to detect the availability of FPGA.

57.18.3.2 STARPU_MAXMAXFPGADEVS

```
#define STARPU_MAXMAXFPGADEVS
```

Define the maximum number of Maxeler FPGA devices that are supported by StarPU.

57.18.4 Function Documentation

57.18.4.1 starpu_max_fpga_get_local_engine()

```
max_engine_t * starpu_max_fpga_get_local_engine (
            void  )
```

Maxeler engine of the current worker. See StarPU/Maxeler FPGA Application for more details.

57.19 Miscellaneous Helpers

Macros

- #define STARPU_MIN(a, b)
- #define STARPU_MAX(a, b)
- #define STARPU_POISON_PTR
- #define starpu_getenv_string_var_default(s, ss, d)
- #define starpu_getenv_size_default(s, d)
- #define starpu_getenv_number(s)
- #define starpu_getenv_number_default(s, d)
- #define starpu_getenv_float_default(s, d)

Functions

- char ∗ starpu_getenv (const char ∗str)
- int starpu_get_env_string_var_default (const char ∗str, const char ∗strings[ ], int defvalue)
- int starpu_get_env_size_default (const char ∗str, int defval)
- static __starpu_inline int starpu_get_env_number (const char ∗str)
- static __starpu_inline int **starpu_get_env_number_default** (const char ∗str, int defval)
- static __starpu_inline float **starpu_get_env_float_default** (const char ∗str, float defval)
- void starpu_execute_on_each_worker (void(∗func)(void ∗), void ∗arg, uint32_t where)
- void starpu_execute_on_each_worker_ex (void(∗func)(void ∗), void ∗arg, uint32_t where, const char ∗name)
- void starpu_execute_on_specific_workers (void(∗func)(void ∗), void ∗arg, unsigned num_workers, unsigned ∗workers, const char ∗name)
- double starpu_timing_now (void)
- int starpu_data_cpy (starpu_data_handle_t dst_handle, starpu_data_handle_t src_handle, int asynchronous, void(∗callback_func)(void ∗), void ∗callback_arg)
- int starpu_data_cpy_priority (starpu_data_handle_t dst_handle, starpu_data_handle_t src_handle, int asynchronous, void(∗callback_func)(void ∗), void ∗callback_arg, int priority)
- int starpu_data_dup_ro (starpu_data_handle_t ∗dst_handle, starpu_data_handle_t src_handle, int asynchronous)
- void starpu_display_bindings (void)
- int starpu_get_pu_os_index (unsigned logical_index)
- long starpu_get_memory_location_bitmap (void ∗ptr, size_t size)
- hwloc_topology_t starpu_get_hwloc_topology (void)

Variables

- int **_starpu_silent**

57.19.1 Detailed Description

57.19.2 Macro Definition Documentation

57.19.2.1 STARPU_MIN

```
#define STARPU_MIN(
                a,
                b )
```

Return the min of the two parameters.

57.19.2.2 STARPU_MAX

```
#define STARPU_MAX(
              a,
              b )
```

Return the max of the two parameters.

57.19.2.3 STARPU_POISON_PTR

```
#define STARPU_POISON_PTR
```

Define a value which can be used to mark pointers as invalid values.

57.19.2.4 starpu_getenv_string_var_default

```
#define starpu_getenv_string_var_default(
              s,
              ss,
              d )
```

Same as starpu_get_env_string_var_default()

57.19.2.5 starpu_getenv_size_default

```
#define starpu_getenv_size_default(
              s,
              d )
```

Same as starpu_get_env_size_default()

57.19.2.6 starpu_getenv_number

```
#define starpu_getenv_number(
              s )
```

Same as starpu_get_env_number()

57.19.2.7 starpu_getenv_number_default

```
#define starpu_getenv_number_default(
              s,
              d )
```

Same as starpu_get_env_number_default()

57.19.2.8 starpu_getenv_float_default

```
#define starpu_getenv_float_default(
              s,
              d )
```

Same as starpu_get_env_float_default()

57.19.3 Function Documentation

57.19.3.1 starpu_getenv()

```
char * starpu_getenv (
             const char * str )
```

Retrieve the value of an environment variable. See Execution Configuration Through Environment Variables for more details.

57.19.3.2 starpu_get_env_string_var_default()

```
int starpu_get_env_string_var_default (
            const char * str,
            const char * strings[],
            int defvalue )
```

If the environment variable `str` is defined and its value is contained in the array `strings`, return the array position. Raise an error if the environment variable `str` is defined with a value not in `strings` Return `defvalue` if the environment variable `str` is not defined. See Execution Configuration Through Environment Variables for more details.

57.19.3.3 starpu_get_env_size_default()

```
int starpu_get_env_size_default (
            const char * str,
            int defval )
```

If the environment variable `str` is defined with a well-defined size value, return the value as a size in bytes. Expected size qualifiers are b, B, k, K, m, M, g, G. The default qualifier is K. If the environment variable `str` is not defined or is empty, return `defval` Raise an error if the value of the environment variable `str` is not well-defined. See Execution Configuration Through Environment Variables for more details.

57.19.3.4 starpu_get_env_number()

```
static __starpu_inline int starpu_get_env_number (
            const char * str )  [static]
```

Return the integer value of the environment variable named `str`. Return 0 otherwise (the variable does not exist or has a non-integer value).

57.19.3.5 starpu_execute_on_each_worker()

```
void starpu_execute_on_each_worker (
            void(*)(void *) func,
            void * arg,
            uint32_t where )
```

Execute the given function `func` on a subset of workers. When calling this method, the offloaded function `func` is executed by every StarPU worker that are eligible to execute the function. The argument `arg` is passed to the offloaded function. The argument `where` specifies on which types of processing units the function should be executed. Similarly to the field starpu_codelet::where, it is possible to specify that the function should be executed on every CUDA device and every CPU by passing STARPU_CPU|STARPU_CUDA. This function blocks until `func` has been executed on every appropriate processing units, and thus may not be called from a callback function for instance. See How To Initialize A Computation Library Once For Each Worker? for more details.

57.19.3.6 starpu_execute_on_each_worker_ex()

```
void starpu_execute_on_each_worker_ex (
            void(*)(void *) func,
            void * arg,
            uint32_t where,
            const char * name )
```

Same as starpu_execute_on_each_worker(), except that the task name is specified in the argument `name`. See How To Initialize A Computation Library Once For Each Worker? for more details.

57.19.3.7 starpu_execute_on_specific_workers()

```
void starpu_execute_on_specific_workers (
            void(*)(void *) func,
            void * arg,
            unsigned num_workers,
            unsigned * workers,
            const char * name )
```

Call `func(arg)` on every worker in the `workers` array. `num_workers` indicates the number of workers in this array. This function is synchronous, but the different workers may execute the function in parallel. See How To Initialize A Computation Library Once For Each Worker? for more details.

57.19.3.8 starpu_timing_now()

```
double starpu_timing_now (
            void )
```

Return the current date in micro-seconds. See Preparing Your Application For Simulation for more details.

57.19.3.9 starpu_data_cpy()

```
int starpu_data_cpy (
            starpu_data_handle_t dst_handle,
            starpu_data_handle_t src_handle,
            int asynchronous,
            void(*)(void *) callback_func,
            void * callback_arg )
```

Copy the content of `src_handle` into `dst_handle`. The parameter `asynchronous` indicates whether the function should block or not. In the case of an asynchronous call, it is possible to synchronize with the termination of this operation either by the means of implicit dependencies (if enabled) or by calling starpu_task_wait_for_all(). If `callback_func` is not `NULL`, this callback function is executed after the handle has been copied, and it is given the pointer `callback_arg` as argument. See Data handles helpers for more details.

57.19.3.10 starpu_data_cpy_priority()

```
int starpu_data_cpy_priority (
            starpu_data_handle_t dst_handle,
            starpu_data_handle_t src_handle,
            int asynchronous,
            void(*)(void *) callback_func,
            void * callback_arg,
            int priority )
```

Like starpu_data_cpy(), copy the content of `src_handle` into `dst_handle`, but additionally take a `priority` parameter to sort it among the whole task graph. See Data handles helpers for more details.

57.19.3.11 starpu_data_dup_ro()

```
int starpu_data_dup_ro (
            starpu_data_handle_t * dst_handle,
            starpu_data_handle_t src_handle,
            int asynchronous )
```

Create a copy of `src_handle`, and return a new handle in `dst_handle`, which is to be used only for read accesses. This allows StarPU to optimize it by not actually copying the data whenever possible (e.g. it may possibly simply return src_handle itself). The parameter `asynchronous` indicates whether the function should block or not. In the case of an asynchronous call, it is possible to synchronize with the termination of this operation either by the means of implicit dependencies (if enabled) or by calling starpu_task_wait_for_all(). If `callback_func` is not `NULL`, this callback function is executed after the handle has been copied, and it is given the pointer `callback_arg` as argument. See Data handles helpers for more details.

57.19.3.12 starpu_display_bindings()

```
void starpu_display_bindings (
            void )
```

Call hwloc-ps to display binding of each process and thread running on the machine.
Use the environment variable STARPU_DISPLAY_BINDINGS to automatically call this function at the beginning of the execution of StarPU. See Miscellaneous And Debug for more details.

57.19.3.13 starpu_get_pu_os_index()

```
int starpu_get_pu_os_index (
            unsigned logical_index )
```

If `hwloc` is used, convert the given `logical_index` of a PU to the OS index of this PU. If `hwloc` is not used, return `logical_index`. See Hardware Topology for more details.

57.19.3.14 starpu_get_memory_location_bitmap()

```
long starpu_get_memory_location_bitmap (
            void * ptr,
            size_t size )
```

Return a bitmap representing logical indexes of NUMA nodes where the buffer targeted by `ptr` is allocated. An error is notified by a negative result. See Hardware Topology for more details.

57.19.3.15 starpu_get_hwloc_topology()

```
hwloc_topology_t starpu_get_hwloc_topology (
            void )
```

Get the hwloc topology used by StarPU. One can use this pointer to get information about topology, but not to change settings related to topology. See Hardware Topology for more details.

57.20 Modularized Scheduler Interface

Data Structures

- struct starpu_sched_component
- struct starpu_sched_tree
- struct starpu_sched_component_fifo_data
- struct starpu_sched_component_prio_data
- struct starpu_sched_component_mct_data
- struct starpu_sched_component_heteroprio_data
- struct starpu_sched_component_perfmodel_select_data
- struct starpu_sched_component_specs

Macros

- #define STARPU_SCHED_COMPONENT_IS_HOMOGENEOUS(component)
- #define STARPU_SCHED_COMPONENT_IS_SINGLE_MEMORY_NODE(component)
- #define **STARPU_COMPONENT_MUTEX_LOCK**(m)
- #define **STARPU_COMPONENT_MUTEX_TRYLOCK**(m)
- #define **STARPU_COMPONENT_MUTEX_UNLOCK**(m)

Enumerations

- enum starpu_sched_component_properties { STARPU_SCHED_COMPONENT_HOMOGENEOUS , STARPU_SCHED_COMPONENT_SINGLE_MEMORY_NODE }

Generic Scheduling Component API

- typedef struct starpu_sched_component ∗(∗ **starpu_sched_component_create_t**) (struct starpu_sched_tree ∗tree, void ∗data)
- struct starpu_sched_component ∗ starpu_sched_component_create (struct starpu_sched_tree ∗tree, const char ∗name) STARPU_ATTRIBUTE_MALLOC
- void starpu_sched_component_destroy (struct starpu_sched_component ∗component)
- void starpu_sched_component_destroy_rec (struct starpu_sched_component ∗component)
- void **starpu_sched_component_add_child** (struct starpu_sched_component ∗component, struct starpu_sched_component ∗child)
- int starpu_sched_component_can_execute_task (struct starpu_sched_component ∗component, struct starpu_task ∗task)
- int starpu_sched_component_execute_preds (struct starpu_sched_component ∗component, struct starpu_task ∗task, double ∗length)
- double starpu_sched_component_transfer_length (struct starpu_sched_component ∗component, struct starpu_task ∗task)
- void **starpu_sched_component_prefetch_on_node** (struct starpu_sched_component ∗component, struct starpu_task ∗task)

Scheduling Tree API

- struct starpu_sched_tree ∗ starpu_sched_tree_create (unsigned sched_ctx_id) STARPU_ATTRIBUTE_MALLOC
- void starpu_sched_tree_destroy (struct starpu_sched_tree ∗tree)
- void starpu_sched_tree_deinitialize (unsigned sched_ctx_id)
- struct starpu_sched_tree ∗ starpu_sched_tree_get (unsigned sched_ctx_id)
- void starpu_sched_tree_update_workers (struct starpu_sched_tree ∗t)
- void starpu_sched_tree_update_workers_in_ctx (struct starpu_sched_tree ∗t)
- int starpu_sched_tree_push_task (struct starpu_task ∗task)
- struct starpu_task ∗ starpu_sched_tree_pop_task (unsigned sched_ctx)
- int starpu_sched_component_push_task (struct starpu_sched_component ∗from, struct starpu_sched_component ∗to, struct starpu_task ∗task)

- struct starpu_task ∗ starpu_sched_component_pull_task (struct starpu_sched_component ∗from, struct starpu_sched_component ∗to)
- struct starpu_task ∗ **starpu_sched_component_pump_to** (struct starpu_sched_component ∗component, struct starpu_sched_component ∗to, int ∗success)
- struct starpu_task ∗ **starpu_sched_component_pump_downstream** (struct starpu_sched_component ∗component, int ∗success)
- int **starpu_sched_component_send_can_push_to_parents** (struct starpu_sched_component ∗component)
- void starpu_sched_tree_add_workers (unsigned sched_ctx_id, int ∗workerids, unsigned nworkers)
- void starpu_sched_tree_remove_workers (unsigned sched_ctx_id, int ∗workerids, unsigned nworkers)
- void starpu_sched_tree_do_schedule (unsigned sched_ctx_id)
- void starpu_sched_component_connect (struct starpu_sched_component ∗parent, struct starpu_sched_component ∗child)

Worker Component API

- struct starpu_sched_component ∗ starpu_sched_component_worker_get (unsigned sched_ctx, int workerid)
- struct starpu_sched_component ∗ **starpu_sched_component_worker_new** (unsigned sched_ctx, int workerid)
- struct starpu_sched_component ∗ starpu_sched_component_parallel_worker_create (struct starpu_sched_tree ∗tree, unsigned nworkers, unsigned ∗workers)
- int starpu_sched_component_worker_get_workerid (struct starpu_sched_component ∗worker_component)
- int starpu_sched_component_is_worker (struct starpu_sched_component ∗component)
- int starpu_sched_component_is_simple_worker (struct starpu_sched_component ∗component)
- int starpu_sched_component_is_combined_worker (struct starpu_sched_component ∗component)
- void starpu_sched_component_worker_pre_exec_hook (struct starpu_task ∗task, unsigned sched_ctx_id)
- void starpu_sched_component_worker_post_exec_hook (struct starpu_task ∗task, unsigned sched_ctx_id)

Flow-control Fifo Component API

These can be used as methods of components. Note: they are not to be called directly, one should really call the methods of the components.

- struct starpu_task ∗ starpu_sched_component_parents_pull_task (struct starpu_sched_component ∗component, struct starpu_sched_component ∗to)
- int starpu_sched_component_can_push (struct starpu_sched_component ∗component, struct starpu_sched_component ∗to)
- int starpu_sched_component_can_pull (struct starpu_sched_component ∗component)
- int starpu_sched_component_can_pull_all (struct starpu_sched_component ∗component)
- double starpu_sched_component_estimated_load (struct starpu_sched_component ∗component)
- double starpu_sched_component_estimated_end_min (struct starpu_sched_component ∗component)
- double starpu_sched_component_estimated_end_min_add (struct starpu_sched_component ∗component, double exp_len)
- double starpu_sched_component_estimated_end_average (struct starpu_sched_component ∗component)
- struct starpu_sched_component ∗ starpu_sched_component_fifo_create (struct starpu_sched_tree ∗tree, struct starpu_sched_component_fifo_data ∗fifo_data) STARPU_ATTRIBUTE_MALLOC
- int starpu_sched_component_is_fifo (struct starpu_sched_component ∗component)

Flow-control Prio Component API

- struct starpu_sched_component ∗ **starpu_sched_component_prio_create** (struct starpu_sched_tree ∗tree, struct starpu_sched_component_prio_data ∗prio_data) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_prio** (struct starpu_sched_component ∗component)

Resource-mapping Work-Stealing Component API

- struct starpu_sched_component ∗ starpu_sched_component_work_stealing_create (struct starpu_sched_tree ∗tree, void ∗arg) STARPU_ATTRIBUTE_MALLOC
- int starpu_sched_component_is_work_stealing (struct starpu_sched_component ∗component)
- int starpu_sched_tree_work_stealing_push_task (struct starpu_task ∗task)

Resource-mapping Random Component API

- struct starpu_sched_component ∗ starpu_sched_component_random_create (struct starpu_sched_tree ∗tree, void ∗arg) STARPU_ATTRIBUTE_MALLOC
- int starpu_sched_component_is_random (struct starpu_sched_component ∗)

Resource-mapping Eager Component API

- struct starpu_sched_component ∗ **starpu_sched_component_eager_create** (struct starpu_sched_tree ∗tree, void ∗arg) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_eager** (struct starpu_sched_component ∗)

Resource-mapping Eager Prio Component API

- struct starpu_sched_component ∗ **starpu_sched_component_eager_prio_create** (struct starpu_sched_tree ∗tree, void ∗arg) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_eager_prio** (struct starpu_sched_component ∗)

Resource-mapping Eager-Calibration Component API

- struct starpu_sched_component ∗ **starpu_sched_component_eager_calibration_create** (struct starpu_sched_tree ∗tree, void ∗arg) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_eager_calibration** (struct starpu_sched_component ∗)

Resource-mapping MCT Component API

- struct starpu_sched_component ∗ starpu_sched_component_mct_create (struct starpu_sched_tree ∗tree, struct starpu_sched_component_mct_data ∗mct_data) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_mct** (struct starpu_sched_component ∗component)

Resource-mapping Heft Component API

- struct starpu_sched_component ∗ **starpu_sched_component_heft_create** (struct starpu_sched_tree ∗tree, struct starpu_sched_component_mct_data ∗mct_data) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_heft** (struct starpu_sched_component ∗component)

Resource-mapping Heteroprio Component API

- struct starpu_sched_component ∗ **starpu_sched_component_heteroprio_create** (struct starpu_sched_tree ∗tree, struct starpu_sched_component_heteroprio_data ∗params) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_heteroprio** (struct starpu_sched_component ∗component)

Special-purpose Best_Implementation Component API

- struct starpu_sched_component ∗ starpu_sched_component_best_implementation_create (struct starpu_sched_tree ∗tree, void ∗arg) STARPU_ATTRIBUTE_MALLOC

Special-purpose Perfmodel_Select Component API

- struct starpu_sched_component ∗ **starpu_sched_component_perfmodel_select_create** (struct starpu_sched_tree ∗tree, struct starpu_sched_component_perfmodel_select_data ∗perfmodel_select_data) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_perfmodel_select** (struct starpu_sched_component ∗component)

Staged pull Component API

- struct starpu_sched_component ∗ **starpu_sched_component_stage_create** (struct starpu_sched_tree ∗tree, void ∗arg) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_stage** (struct starpu_sched_component ∗component)

User-choice push Component API

- struct starpu_sched_component ∗ **starpu_sched_component_userchoice_create** (struct starpu_sched_tree ∗tree, void ∗arg) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_userchoice** (struct starpu_sched_component ∗component)

Recipe Component API

- struct starpu_sched_component_composed_recipe ∗ starpu_sched_component_composed_recipe_create (void) STARPU_ATTRIBUTE_MALLOC
- struct starpu_sched_component_composed_recipe ∗ starpu_sched_component_composed_recipe_create_singleton (struct starpu_sched_component ∗(∗create_component)(struct starpu_sched_tree ∗tree, void ∗arg), void ∗arg) STARPU_ATTRIBUTE_MALLOC
- void starpu_sched_component_composed_recipe_add (struct starpu_sched_component_composed_recipe ∗recipe, struct starpu_sched_component ∗(∗create_component)(struct starpu_sched_tree ∗tree, void ∗arg), void ∗arg)
- void starpu_sched_component_composed_recipe_destroy (struct starpu_sched_component_composed_recipe ∗)
- struct starpu_sched_component ∗ starpu_sched_component_composed_component_create (struct starpu_sched_tree ∗tree, struct starpu_sched_component_composed_recipe ∗recipe) STARPU_ATTRIBUTE_MALLOC
- struct starpu_sched_tree ∗ starpu_sched_component_make_scheduler (unsigned sched_ctx_id, struct starpu_sched_component_specs s)

Basic API

- void starpu_sched_component_initialize_simple_scheduler (starpu_sched_component_create_t create_decision_component, void ∗data, unsigned flags, unsigned sched_ctx_id)
- void starpu_sched_component_initialize_simple_schedulers (unsigned sched_ctx_id, unsigned ndecisions,...)
- #define **STARPU_SCHED_SIMPLE_DECIDE_MASK**
- #define STARPU_SCHED_SIMPLE_DECIDE_WORKERS
- #define STARPU_SCHED_SIMPLE_DECIDE_MEMNODES
- #define STARPU_SCHED_SIMPLE_DECIDE_ARCHS
- #define STARPU_SCHED_SIMPLE_DECIDE_ALWAYS
- #define STARPU_SCHED_SIMPLE_PERFMODEL
- #define STARPU_SCHED_SIMPLE_IMPL
- #define STARPU_SCHED_SIMPLE_FIFO_ABOVE
- #define STARPU_SCHED_SIMPLE_FIFO_ABOVE_PRIO
- #define STARPU_SCHED_SIMPLE_FIFOS_BELOW
- #define STARPU_SCHED_SIMPLE_FIFOS_BELOW_PRIO
- #define STARPU_SCHED_SIMPLE_FIFOS_BELOW_READY
- #define STARPU_SCHED_SIMPLE_WS_BELOW
- #define STARPU_SCHED_SIMPLE_COMBINED_WORKERS
- #define STARPU_SCHED_SIMPLE_FIFOS_BELOW_EXP
- #define STARPU_SCHED_SIMPLE_PRE_DECISION

57.20.1 Detailed Description

57.20.2 Data Structure Documentation

57.20.2.1 struct starpu_sched_component

Structure for a scheduler module. A scheduler is a tree-like structure of them, some parts of scheduler can be shared by several contexes to perform some local optimisations, so, for all components, a list of parent is defined by `sched_ctx_id`. They embed there specialised method in a pseudo object-style, so calls are like `component->push_task(component,task)`

Data Fields

- struct starpu_sched_tree ∗ tree
- struct starpu_bitmap workers
- struct starpu_bitmap workers_in_ctx
- void ∗ data
- char ∗ **name**
- unsigned nchildren
- struct starpu_sched_component ∗∗ children
- unsigned nparents
- struct starpu_sched_component ∗∗ parents
- void(∗ add_child)(struct starpu_sched_component ∗component, struct starpu_sched_component ∗child)
- void(∗ remove_child)(struct starpu_sched_component ∗component, struct starpu_sched_component ∗child)
- void(∗ **add_parent**)(struct starpu_sched_component ∗component, struct starpu_sched_component ∗parent)
- void(∗ **remove_parent**)(struct starpu_sched_component ∗component, struct starpu_sched_component ∗parent)
- int(∗ push_task)(struct starpu_sched_component ∗, struct starpu_task ∗)
- struct starpu_task ∗(∗ pull_task)(struct starpu_sched_component ∗from, struct starpu_sched_component ∗to)
- int(∗ can_push)(struct starpu_sched_component ∗from, struct starpu_sched_component ∗to)
- int(∗ can_pull)(struct starpu_sched_component ∗component)
- void(∗ do_schedule)(struct starpu_sched_component ∗component)
- int(∗ **notify**)(struct starpu_sched_component ∗component, int message_ID, void ∗arg)
- double(∗ estimated_load)(struct starpu_sched_component ∗component)
- double(∗ estimated_end)(struct starpu_sched_component ∗component)
- void(∗ deinit_data)(struct starpu_sched_component ∗component)
- void(∗ notify_change_workers)(struct starpu_sched_component ∗component)
- int **properties**
- hwloc_obj_t obj

57.20.2.1.1 Field Documentation

57.20.2.1.1.1 tree `struct starpu_sched_tree* starpu_sched_component::tree`

The tree containing the component

57.20.2.1.1.2 workers `struct starpu_bitmap starpu_sched_component::workers`

set of underlying workers

57.20.2.1.1.3 workers_in_ctx `struct starpu_bitmap starpu_sched_component::workers_in_ctx`

subset of starpu_sched_component::workers that is currently available in the context The push method should take this value into account, it is set with: component->workers UNION tree->workers UNION component->child[i]->workers_in_ctx iff exist x such as component->children[i]->parents[x] == component

57.20.2.1.1.4 data `void* starpu_sched_component::data`

private data

57.20.2.1.1.5 nchildren `unsigned starpu_sched_component::nchildren`

number of compoments's children

57.20.2.1.1.6 children `struct starpu_sched_component** starpu_sched_component::children`

vector of component's children

57.20.2.1.1.7 nparents `unsigned starpu_sched_component::nparents`

number of component's parents

57.20.2.1.1.8 parents

`struct starpu_sched_component** starpu_sched_component::parents`

vector of component's parents

57.20.2.1.1.9 add_child

`void(* starpu_sched_component::add_child) (struct starpu_sched_component *component, struct starpu_sched_component *child)`

add a child to component

57.20.2.1.1.10 remove_child

`void(* starpu_sched_component::remove_child) (struct starpu_sched_component *component, struct starpu_sched_component *child)`

remove a child from component

57.20.2.1.1.11 push_task

`int(* starpu_sched_component::push_task) (struct starpu_sched_component *, struct starpu_task *)`

push a task in the scheduler module. this function is called to push a task on component subtree, this can either perform a recursive call on a child or store the task in the component, then it will be returned by a further pull_task call. the caller must ensure that component is able to execute task. This method must either return 0 if it the task was properly stored or passed over to a child component, or return a value different from 0 if the task could not be consumed (e.g. the queue is full).

57.20.2.1.1.12 pull_task

`struct starpu_task *(* starpu_sched_component::pull_task) (struct starpu_sched_compone *from, struct starpu_sched_component *to)`

pop a task from the scheduler module. this function is called by workers to get a task from their parents. this function should first return a locally stored task or perform a recursive call on the parents. the task returned by this function should be executable by the caller

57.20.2.1.1.13 can_push

`int(* starpu_sched_component::can_push) (struct starpu_sched_component *from, struct starpu_sched_component *to)`

This function is called by a component which implements a queue, allowing it to signify to its parents that an empty slot is available in its queue. This should return 1 if some tasks could be pushed The basic implementation of this function is a recursive call to its parents, the user has to specify a personally-made function to catch those calls.

57.20.2.1.1.14 can_pull

`int(* starpu_sched_component::can_pull) (struct starpu_sched_component *component)`

This function allow a component to wake up a worker. It is currently called by component which implements a queue, to signify to its children that a task have been pushed in its local queue, and is available to be popped by a worker, for example. This should return 1 if some some container or worker could (or will) pull some tasks. The basic implementation of this function is a recursive call to its children, until at least one worker have been woken up.

57.20.2.1.1.15 do_schedule

`void(* starpu_sched_component::do_schedule) (struct starpu_sched_component *component)`

This function is called when starpu_do_schedule() is called by the application.

57.20.2.1.1.16 estimated_load

`double(* starpu_sched_component::estimated_load) (struct starpu_sched_component *component)`

heuristic to compute load of scheduler module. Basically the number of tasks divided by the sum of relatives speedup of workers available in context. estimated_load(component) = sum(estimated_load(component_children)) + nb_local_tasks / average(relative_speedup(underlying_worker))

57.20.2.1.1.17 estimated_end

`double(* starpu_sched_component::estimated_end) (struct starpu_sched_component *component)`

return the time when a worker will enter in starvation. This function is relevant only if the task->predicted member has been set.

57.20.2.1.1.18 deinit_data `void(* starpu_sched_component::deinit_data) (struct starpu_sched_component *component)`

called by starpu_sched_component_destroy. Should free data allocated during creation

57.20.2.1.1.19 notify_change_workers `void(* starpu_sched_component::notify_change_workers) (struct starpu_sched_component *component)`

this function is called for each component when workers are added or removed from a context

57.20.2.1.1.20 obj `hwloc_obj_t starpu_sched_component::obj`

the hwloc object associated to scheduler module. points to the part of topology that is binded to this component, eg: a numa node for a ws component that would balance load between underlying sockets

57.20.2.2 struct starpu_sched_tree

The actual scheduler

Data Fields

struct starpu_sched_component ∗	root	entry module of the scheduler
struct starpu_bitmap	workers	set of workers available in this context, this value is used to mask workers in modules
unsigned	sched_ctx_id	context id of the scheduler
starpu_pthread_mutex_t	lock	lock used to protect the scheduler, it is taken in read mode pushing a task and in write mode for adding or removing workers

57.20.2.3 struct starpu_sched_component_fifo_data

todo

Data Fields

unsigned	ntasks_threshold	
double	exp_len_threshold	
int	ready	
int	exp	

57.20.2.4 struct starpu_sched_component_prio_data

todo

Data Fields

unsigned	ntasks_threshold	
double	exp_len_threshold	
int	ready	
int	exp	

57.20.2.5 struct starpu_sched_component_mct_data

todo

Data Fields

double	alpha	
double	beta	
double	_gamma	
double	idle_power	

57.20.2.6 struct starpu_sched_component_heteroprio_data

todo

Data Fields

struct starpu_sched_component_mct_data ∗	mct	
unsigned	batch	

57.20.2.7 struct starpu_sched_component_perfmodel_select_data

todo

Data Fields

struct starpu_sched_component ∗	calibrator_component	
struct starpu_sched_component ∗	no_perfmodel_component	
struct starpu_sched_component ∗	perfmodel_component	

57.20.2.8 struct starpu_sched_component_specs

Define how build a scheduler according to topology. Each level (except for hwloc_machine_composed_sched_component) can be `NULL`, then the level is just skipped. Bugs everywhere, do not rely on.

Data Fields

- struct starpu_sched_component_composed_recipe ∗ hwloc_machine_composed_sched_component
- struct starpu_sched_component_composed_recipe ∗ hwloc_component_composed_sched_component
- struct starpu_sched_component_composed_recipe ∗ hwloc_socket_composed_sched_component
- struct starpu_sched_component_composed_recipe ∗ hwloc_cache_composed_sched_component
- struct starpu_sched_component_composed_recipe ∗(∗ worker_composed_sched_component)(enum starpu_worker_archtype archtype)
- int mix_heterogeneous_workers

57.20.2.8.1 Field Documentation

57.20.2.8.1.1 hwloc_machine_composed_sched_component `struct starpu_sched_component_composed_recipe* starpu_sched_component_specs::hwloc_machine_composed_sched_component`

the composed component to put on the top of the scheduler this member must not be `NULL` as it is the root of the topology

57.20.2.8.1.2 hwloc_component_composed_sched_component `struct starpu_sched_component_composed_recipe* starpu_sched_component_specs::hwloc_component_composed_sched_component`

the composed component to put for each memory component

57.20.2.8.1.3 hwloc_socket_composed_sched_component `struct starpu_sched_component_composed_recipe* starpu_sched_component_specs::hwloc_socket_composed_sched_component`

the composed component to put for each socket

57.20.2.8.1.4 hwloc_cache_composed_sched_component `struct starpu_sched_component_composed_recipe* starpu_sched_component_specs::hwloc_cache_composed_sched_component`

the composed component to put for each cache

57.20.2.8.1.5 worker_composed_sched_component `struct starpu_sched_component_composed_recipe *(* starpu_sched_component_specs::worker_composed_sched_component) (enum starpu_worker_archtype archtype)`

a function that return a starpu_sched_component_composed_recipe to put on top of a worker of type `archtype`. `NULL` is a valid return value, then no component will be added on top

57.20.2.8.1.6 mix_heterogeneous_workers `int starpu_sched_component_specs::mix_heterogeneous_workers`

this flag is a dirty hack because of the poor expressivity of this interface. As example, if you want to build a heft component with a fifo component per numa component, and you also have GPUs, if this flag is set, GPUs will share those fifos. If this flag is not set, a new fifo will be built for each of them (if they have the same starpu_perf_arch and the same numa component it will be shared. it indicates if heterogeneous workers should be brothers or cousins, as example, if a gpu and a cpu should share or not there numa node

57.20.3 Macro Definition Documentation

57.20.3.1 STARPU_SCHED_COMPONENT_IS_HOMOGENEOUS

```
#define STARPU_SCHED_COMPONENT_IS_HOMOGENEOUS(
            component )
```

indicate if component is homogeneous

57.20.3.2 STARPU_SCHED_COMPONENT_IS_SINGLE_MEMORY_NODE

```
#define STARPU_SCHED_COMPONENT_IS_SINGLE_MEMORY_NODE(
            component )
```

indicate if all workers have the same memory component

57.20.3.3 STARPU_SCHED_SIMPLE_DECIDE_WORKERS

```
#define STARPU_SCHED_SIMPLE_DECIDE_WORKERS
```

Request to create downstream queues per worker, i.e. the scheduling decision-making component will choose exactly which workers tasks should got to.

57.20.3.4 STARPU_SCHED_SIMPLE_DECIDE_MEMNODES

```
#define STARPU_SCHED_SIMPLE_DECIDE_MEMNODES
```

Request to create downstream queues per memory nodes, i.e. the scheduling decision-making component will choose which memory node tasks will go to.

57.20.3.5 STARPU_SCHED_SIMPLE_DECIDE_ARCHS

```
#define STARPU_SCHED_SIMPLE_DECIDE_ARCHS
```

Request to create downstream queues per computation arch, i.e. the scheduling decision-making component will choose whether tasks go to CPUs, or CUDA, or OpenCL, etc.

57.20.3.6 STARPU_SCHED_SIMPLE_DECIDE_ALWAYS

`#define STARPU_SCHED_SIMPLE_DECIDE_ALWAYS`

Request to create the scheduling decision-making component even if there is only one available choice. This is useful for instance when the decision-making component will store tasks itself (and not use STARPU_SCHED_SIMPLE_FIFO_ABOVE) to decide in which order tasks should be passed below.

57.20.3.7 STARPU_SCHED_SIMPLE_PERFMODEL

`#define STARPU_SCHED_SIMPLE_PERFMODEL`

Request to add a perfmodel selector above the scheduling decision-making component. That way, only tasks with a calibrated performance model will be given to the component, other tasks will go to an eager branch that will distributed tasks so that their performance models will get calibrated. In other words, this is needed when using a component which needs performance models for tasks.

57.20.3.8 STARPU_SCHED_SIMPLE_IMPL

`#define STARPU_SCHED_SIMPLE_IMPL`

Request that a component be added just above workers, that chooses the best task implementation.

57.20.3.9 STARPU_SCHED_SIMPLE_FIFO_ABOVE

`#define STARPU_SCHED_SIMPLE_FIFO_ABOVE`

Request to create a fifo above the scheduling decision-making component, otherwise tasks will be pushed directly to the component.
This is useful to store tasks if there is a fifo below which limits the number of tasks to be scheduld in advance. The scheduling decision-making component can also store tasks itself, in which case this flag is not useful.

57.20.3.10 STARPU_SCHED_SIMPLE_FIFO_ABOVE_PRIO

`#define STARPU_SCHED_SIMPLE_FIFO_ABOVE_PRIO`

Request that the fifo above be sorted by priorities

57.20.3.11 STARPU_SCHED_SIMPLE_FIFOS_BELOW

`#define STARPU_SCHED_SIMPLE_FIFOS_BELOW`

Request to create fifos below the scheduling decision-making component, otherwise tasks will be pulled directly from workers.
This is useful to be able to schedule a (tunable) small number of tasks in advance only.

57.20.3.12 STARPU_SCHED_SIMPLE_FIFOS_BELOW_PRIO

`#define STARPU_SCHED_SIMPLE_FIFOS_BELOW_PRIO`

Request that the fifos below be sorted by priorities

57.20.3.13 STARPU_SCHED_SIMPLE_FIFOS_BELOW_READY

`#define STARPU_SCHED_SIMPLE_FIFOS_BELOW_READY`

Request that the fifos below be pulled rather ready tasks

57.20.3.14 STARPU_SCHED_SIMPLE_WS_BELOW

`#define STARPU_SCHED_SIMPLE_WS_BELOW`

Request that work between workers using the same fifo below be distributed using a work stealing component.

57.20.3.15 STARPU_SCHED_SIMPLE_COMBINED_WORKERS

`#define STARPU_SCHED_SIMPLE_COMBINED_WORKERS`

Request to not only choose between simple workers, but also choose between combined workers.

57.20.3.16 STARPU_SCHED_SIMPLE_FIFOS_BELOW_EXP

`#define STARPU_SCHED_SIMPLE_FIFOS_BELOW_EXP`

Request that the fifos below keep track of expected duration, start and end time of theirs elements

57.20.3.17 STARPU_SCHED_SIMPLE_PRE_DECISION

`#define STARPU_SCHED_SIMPLE_PRE_DECISION`

Request to prepend a component before the decision component. This should be used alone and followed by the component creation function pointer and its data.

57.20.4 Enumeration Type Documentation

57.20.4.1 starpu_sched_component_properties

`enum starpu_sched_component_properties`

flags for starpu_sched_component::properties

Enumerator

STARPU_SCHED_COMPONENT_HOMOGENEOUS	indicate that all workers have the same starpu_worker_archtype
STARPU_SCHED_COMPONENT_SINGLE_MEMORY_NODE	indicate that all workers have the same memory component

57.20.5 Function Documentation

57.20.5.1 starpu_sched_tree_create()

```
struct starpu_sched_tree * starpu_sched_tree_create (
            unsigned sched_ctx_id )
```

create a empty initialized starpu_sched_tree. See Implementing a Modularized Scheduler for more details.

57.20.5.2 starpu_sched_tree_destroy()

```
void starpu_sched_tree_destroy (
            struct starpu_sched_tree * tree )
```

destroy tree and free all non shared component in it. See Implementing a Modularized Scheduler for more details.

57.20.5.3 starpu_sched_tree_deinitialize()

```
void starpu_sched_tree_deinitialize (
            unsigned sched_ctx_id )
```

calls starpu_sched_tree_destroy, ready for use for starpu_sched_policy::deinit_sched field. See Implementing a Modularized Schedul
for more details.

57.20.5.4 starpu_sched_tree_get()

```
struct starpu_sched_tree * starpu_sched_tree_get (
            unsigned sched_ctx_id )
```

See Implementing a Modularized Scheduler for more details.

57.20.5.5 starpu_sched_tree_update_workers()

```
void starpu_sched_tree_update_workers (
            struct starpu_sched_tree * t )
```

recursively set all starpu_sched_component::workers, do not take into account shared parts (except workers). See Implementing a Modularized Scheduler for more details.

57.20.5.6 starpu_sched_tree_update_workers_in_ctx()

```
void starpu_sched_tree_update_workers_in_ctx (
            struct starpu_sched_tree * t )
```

recursively set all starpu_sched_component::workers_in_ctx, do not take into account shared parts (except workers) See Implementing a Modularized Scheduler for more details.

57.20.5.7 starpu_sched_tree_push_task()

```
int starpu_sched_tree_push_task (
            struct starpu_task * task )
```

compatibility with starpu_sched_policy interface. See Implementing a Modularized Scheduler for more details.

57.20.5.8 starpu_sched_tree_pop_task()

```
struct starpu_task * starpu_sched_tree_pop_task (
            unsigned sched_ctx )
```

compatibility with starpu_sched_policy interface. See Implementing a Modularized Scheduler for more details.

57.20.5.9 starpu_sched_component_push_task()

```
int starpu_sched_component_push_task (
            struct starpu_sched_component * from,
            struct starpu_sched_component * to,
            struct starpu_task * task )
```

Push a task to a component. This is a helper for `component->push_task(component, task)` plus tracing.

57.20.5.10 starpu_sched_component_pull_task()

```
struct starpu_task * starpu_sched_component_pull_task (
            struct starpu_sched_component * from,
            struct starpu_sched_component * to )
```

Pull a task from a component. This is a helper for `component->pull_task(component)` plus tracing.

57.20.5.11 starpu_sched_tree_add_workers()

```
void starpu_sched_tree_add_workers (
            unsigned sched_ctx_id,
            int * workerids,
            unsigned nworkers )
```

compatibility with starpu_sched_policy interface

57.20.5.12 starpu_sched_tree_remove_workers()

```
void starpu_sched_tree_remove_workers (
            unsigned sched_ctx_id,
            int * workerids,
            unsigned nworkers )
```

compatibility with starpu_sched_policy interface

57.20.5.13 starpu_sched_tree_do_schedule()

```
void starpu_sched_tree_do_schedule (
            unsigned sched_ctx_id )
```

Run the do_schedule method of the components. This is a helper for starpu_sched_policy::do_schedule.

57.20.5.14 starpu_sched_component_connect()

```
void starpu_sched_component_connect (
            struct starpu_sched_component * parent,
            struct starpu_sched_component * child )
```

Attach component `child` to parent `parent`. Some component may accept only one child, others accept several (e.g. MCT)

57.20.5.15 starpu_sched_component_create()

```
struct starpu_sched_component * starpu_sched_component_create (
            struct starpu_sched_tree * tree,
            const char * name )
```

allocate and initialize component field with defaults values : .pop_task make recursive call on father .estimated_load compute relative speedup and tasks in sub tree .estimated_end return the minimum of recursive call on children .add_child is starpu_sched_component_add_child .remove_child is starpu_sched_component_remove_child .notify_change_workers does nothing .deinit_data does nothing

57.20.5.16 starpu_sched_component_destroy()

```
void starpu_sched_component_destroy (
            struct starpu_sched_component * component )
```

free data allocated by starpu_sched_component_create and call component->deinit_data(component) set to `NULL` the member starpu_sched_component::fathers[sched_ctx_id] of all child if its equal to `component`

57.20.5.17 starpu_sched_component_destroy_rec()

```
void starpu_sched_component_destroy_rec (
            struct starpu_sched_component * component )
```

recursively destroy non shared parts of a `component` 's tree

57.20.5.18 starpu_sched_component_can_execute_task()

```
int starpu_sched_component_can_execute_task (
            struct starpu_sched_component * component,
            struct starpu_task * task )
```

return true iff `component` can execute `task`, this function take into account the workers available in the scheduling context

57.20.5.19 starpu_sched_component_execute_preds()

```
int starpu_sched_component_execute_preds (
            struct starpu_sched_component * component,
            struct starpu_task * task,
            double * length )
```

return a non `NULL` value if `component` can execute `task`. write the execution prediction length for the best implementation of the best worker available and write this at `length` address. this result is more relevant if starpu_sched_component::is_homogeneous is non `NULL`. if a worker need to be calibrated for an implementation, nan is set to `length`.

57.20.5.20 starpu_sched_component_transfer_length()

```
double starpu_sched_component_transfer_length (
            struct starpu_sched_component * component,
            struct starpu_task * task )
```

return the average time to transfer `task` data to underlying `component` workers.

57.20.5.21 starpu_sched_component_worker_get()

```
struct starpu_sched_component * starpu_sched_component_worker_get (
            unsigned sched_ctx,
            int workerid )
```

return the struct starpu_sched_component corresponding to `workerid`. Undefined if `workerid` is not a valid workerid

57.20.5.22 starpu_sched_component_parallel_worker_create()

```
struct starpu_sched_component * starpu_sched_component_parallel_worker_create (
            struct starpu_sched_tree * tree,
            unsigned nworkers,
            unsigned * workers )
```

Create a combined worker that pushes tasks in parallel to workers `workers` (size `nworkers`).

57.20.5.23 starpu_sched_component_worker_get_workerid()

```
int starpu_sched_component_worker_get_workerid (
            struct starpu_sched_component * worker_component )
```

return the workerid of `worker_component`, undefined if starpu_sched_component_is_worker(worker_component) == 0

57.20.5.24 starpu_sched_component_is_worker()

```
int starpu_sched_component_is_worker (
            struct starpu_sched_component * component )
```

return true iff `component` is a worker component

57.20.5.25 starpu_sched_component_is_simple_worker()

```
int starpu_sched_component_is_simple_worker (
            struct starpu_sched_component * component )
```

return true iff `component` is a simple worker component

57.20.5.26 starpu_sched_component_is_combined_worker()

```
int starpu_sched_component_is_combined_worker (
            struct starpu_sched_component * component )
```

return true iff `component` is a combined worker component

57.20.5.27 starpu_sched_component_worker_pre_exec_hook()

```
void starpu_sched_component_worker_pre_exec_hook (
            struct starpu_task * task,
            unsigned sched_ctx_id )
```

compatibility with starpu_sched_policy interface update predictions for workers

57.20.5.28 starpu_sched_component_worker_post_exec_hook()

```
void starpu_sched_component_worker_post_exec_hook (
            struct starpu_task * task,
            unsigned sched_ctx_id )
```

compatibility with starpu_sched_policy interface

57.20.5.29 starpu_sched_component_parents_pull_task()

```
struct starpu_task * starpu_sched_component_parents_pull_task (
            struct starpu_sched_component * component,
            struct starpu_sched_component * to )
```

default function for the pull component method, just call pull of parents until one of them returns a task

57.20.5.30 starpu_sched_component_can_push()

```
int starpu_sched_component_can_push (
            struct starpu_sched_component * component,
            struct starpu_sched_component * to )
```

default function for the can_push component method, just call can_push of parents until one of them returns non-zero

57.20.5.31 starpu_sched_component_can_pull()

```
int starpu_sched_component_can_pull (
            struct starpu_sched_component * component )
```

default function for the can_pull component method, just call can_pull of children until one of them returns non-zero

57.20.5.32 starpu_sched_component_can_pull_all()

```
int starpu_sched_component_can_pull_all (
            struct starpu_sched_component * component )
```

function for the can_pull component method, call can_pull of all children

57.20.5.33 starpu_sched_component_estimated_load()

```
double starpu_sched_component_estimated_load (
            struct starpu_sched_component * component )
```

default function for the estimated_load component method, just sum up the loads of the children of the component.

57.20.5.34 starpu_sched_component_estimated_end_min()

```
double starpu_sched_component_estimated_end_min (
            struct starpu_sched_component * component )
```

function that can be used for the estimated_end component method, compute the minimum completion time of the children.

57.20.5.35 starpu_sched_component_estimated_end_min_add()

```
double starpu_sched_component_estimated_end_min_add (
            struct starpu_sched_component * component,
            double exp_len )
```

function that can be used for the estimated_end component method, compute the minimum completion time of the children, and add to it an estimation of how existing queued work, plus the exp_len work, can be completed. This is typically used instead of starpu_sched_component_estimated_end_min when the component contains a queue of tasks, which thus needs to be added to the estimations.

57.20.5.36 starpu_sched_component_estimated_end_average()

```
double starpu_sched_component_estimated_end_average (
            struct starpu_sched_component * component )
```

default function for the estimated_end component method, compute the average completion time of the children.

57.20.5.37 starpu_sched_component_fifo_create()

```
struct starpu_sched_component * starpu_sched_component_fifo_create (
            struct starpu_sched_tree * tree,
            struct starpu_sched_component_fifo_data * fifo_data )
```

Return a struct starpu_sched_component with a fifo. A stable sort is performed according to tasks priorities. A push_task call on this component does not perform recursive calls, underlying components will have to call pop_task to get it. starpu_sched_component::estimated_end function compute the estimated length by dividing the sequential length by the number of underlying workers.

57.20.5.38 starpu_sched_component_is_fifo()

```
int starpu_sched_component_is_fifo (
            struct starpu_sched_component * component )
```

return true iff `component` is a fifo component

57.20.5.39 starpu_sched_component_work_stealing_create()

```
struct starpu_sched_component * starpu_sched_component_work_stealing_create (
            struct starpu_sched_tree * tree,
            void * arg )
```

return a component that perform a work stealing scheduling. Tasks are pushed in a round robin way. estimated_end return the average of expected length of fifos, starting at the average of the expected_end of his children. When a worker have to steal a task, it steal a task in a round robin way, and get the last pushed task of the higher priority.

57.20.5.40 starpu_sched_component_is_work_stealing()

```
int starpu_sched_component_is_work_stealing (
            struct starpu_sched_component * component )
```

return true iff `component` is a work stealing component

57.20.5.41 starpu_sched_tree_work_stealing_push_task()

```
int starpu_sched_tree_work_stealing_push_task (
            struct starpu_task * task )
```

undefined if there is no work stealing component in the scheduler. If any, `task` is pushed in a default way if the caller is the application, and in the caller's fifo if its a worker.

57.20.5.42 starpu_sched_component_random_create()

```
struct starpu_sched_component * starpu_sched_component_random_create (
            struct starpu_sched_tree * tree,
            void * arg )
```

create a component that perform a random scheduling

57.20.5.43 starpu_sched_component_is_random()

```
int starpu_sched_component_is_random (
            struct starpu_sched_component *  )
```

return true iff `component` is a random component

57.20.5.44 starpu_sched_component_mct_create()

```
struct starpu_sched_component * starpu_sched_component_mct_create (
            struct starpu_sched_tree * tree,
            struct starpu_sched_component_mct_data * mct_data )
```

create a component with mct_data parameters. the mct component does not do anything but pushing tasks on no_perf_model_component and calibrating_component

57.20.5.45 starpu_sched_component_best_implementation_create()

```
struct starpu_sched_component * starpu_sched_component_best_implementation_create (
            struct starpu_sched_tree * tree,
            void * arg )
```

Select the implementation that offer the shortest computation length for the first worker that can execute the task. Or an implementation that need to be calibrated. Also set starpu_task::predicted and starpu_task::predicted_transfer for memory component of the first suitable workerid. If starpu_sched_component::push method is called and starpu_sched_component::nchild > 1 the result is undefined.

57.20.5.46 starpu_sched_component_composed_recipe_create()

```
struct starpu_sched_component_composed_recipe * starpu_sched_component_composed_recipe_create
(
            void  )
```

return an empty recipe for a composed component, it should not be used without modification. See Implementing a Modularized Scheduler for more details.

57.20.5.47 starpu_sched_component_composed_recipe_create_singleton()

```
struct starpu_sched_component_composed_recipe * starpu_sched_component_composed_recipe_create↩
_singleton (
            struct starpu_sched_component *(*)(struct starpu_sched_tree *tree, void *arg)
create_component,
            void * arg )
```

return a recipe to build a composed component with a `create_component`

57.20.5.48 starpu_sched_component_composed_recipe_add()

```
void starpu_sched_component_composed_recipe_add (
            struct starpu_sched_component_composed_recipe * recipe,
            struct starpu_sched_component *(*)(struct starpu_sched_tree *tree, void *arg)
create_component,
            void * arg )
```

add `create_component` under all previous components in recipe

57.20.5.49 starpu_sched_component_composed_recipe_destroy()

```
void starpu_sched_component_composed_recipe_destroy (
            struct starpu_sched_component_composed_recipe *  )
```

destroy composed_sched_component, this should be done after starpu_sched_component_composed_↩ component_create was called

57.20.5.50 starpu_sched_component_composed_component_create()

```
struct starpu_sched_component * starpu_sched_component_composed_component_create (
            struct starpu_sched_tree * tree,
            struct starpu_sched_component_composed_recipe * recipe )
```

create a component that behave as all component of recipe where linked. Except that you can not use starpu_↩ sched_component_is_foo function if recipe contain a single create_foo arg_foo pair, create_foo(arg_foo) is returned instead of a composed component

57.20.5.51 starpu_sched_component_make_scheduler()

```
struct starpu_sched_tree * starpu_sched_component_make_scheduler (
            unsigned sched_ctx_id,
            struct starpu_sched_component_specs s )
```

build a scheduler for `sched_ctx_id` according to `s` and the hwloc topology of the machine.

57.20.5.52 starpu_sched_component_initialize_simple_scheduler()

```
void starpu_sched_component_initialize_simple_scheduler (
            starpu_sched_component_create_t create_decision_component,
            void * data,
            unsigned flags,
            unsigned sched_ctx_id )
```

Create a simple modular scheduler tree around a scheduling decision-making component `component`. The details of what should be built around `component` is described by `flags`. The different STARPU_SCHED_SIMPL_DECIDE_∗ flags are mutually exclusive. `data` is passed to the `create_decision_component` function when creating the decision component. See Implementing a Modularized Scheduler for more details.

57.20.5.53 starpu_sched_component_initialize_simple_schedulers()

```
void starpu_sched_component_initialize_simple_schedulers (
            unsigned sched_ctx_id,
            unsigned ndecisions,
             ... )
```

Create a simple modular scheduler tree around several scheduling decision-making components. The parameters are similar to starpu_sched_component_initialize_simple_scheduler, but per scheduling decision, for instance:
starpu_sched_component_initialize_simple_schedulers(sched_ctx_id, 2, create1, data1, flags1, create2, data2, flags2);
The different flags parameters must be coherent: same decision flags. They must not include the perfmodel flag (not supported yet).

57.21 MPI Fault Tolerance Support

Functions

- int starpu_mpi_checkpoint_init (void)
- int starpu_mpi_checkpoint_shutdown (void)
- int starpu_mpi_checkpoint_template_register (starpu_mpi_checkpoint_template_t ∗cp_template, int cp_id, int cp_domain,...)
- int starpu_mpi_checkpoint_template_create (starpu_mpi_checkpoint_template_t ∗cp_template, int cp_id, int cp_domain)
- int starpu_mpi_checkpoint_template_add_entry (starpu_mpi_checkpoint_template_t ∗cp_template,...)
- int starpu_mpi_checkpoint_template_freeze (starpu_mpi_checkpoint_template_t ∗cp_template)
- int starpu_mpi_checkpoint_template_submit (starpu_mpi_checkpoint_template_t cp_template, int prio)
- int **starpu_mpi_checkpoint_template_print** (starpu_mpi_checkpoint_template_t cp_template)

57.21.1 Detailed Description

57.21.2 Function Documentation

57.21.2.1 starpu_mpi_checkpoint_init()

```
int starpu_mpi_checkpoint_init (
            void )
```

Initialise the checkpoint mechanism

57.21.2.2 starpu_mpi_checkpoint_shutdown()

```
int starpu_mpi_checkpoint_shutdown (
            void )
```

Shutdown the checkpoint mechanism

57.21.2.3 starpu_mpi_checkpoint_template_register()

```
int starpu_mpi_checkpoint_template_register (
            starpu_mpi_checkpoint_template_t * cp_template,
            int cp_id,
            int cp_domain,
             ... )
```

Wrapped function to register a checkpoint template `cp_template` with the given arguments. It is then ready to use with starpu_mpi_checkpoint_template_submit() during the program execution. This command executes starpu_mpi_checkpoint_template_create(), adds the given checkpoint entry and freezes the checkpoint, and therefore can no longer be modified. A unique checkpoint id `cp_id` is requested from the user in order to create several templates and to match with a corresponding ::starpu_mpi_init_from_checkpoint() (not implemented yet).

The arguments following the `cp_template` and the `cp_id` can be of the following types:

- STARPU_R followed by a data handle and the backup rank;
- STARPU_DATA_ARRAY followed by an array of data handles, its number of elements and a backup rank (non functional);
- STARPU_VALUE followed by a pointer to the unregistered value, its size in bytes, a unique tag (as the ones given for data handle registering) and the function giving the back up rank of the rank argument : int(backup↩_of)(int) .
- The argument list must be ended by the value 0.

57.21.2.4 starpu_mpi_checkpoint_template_create()

```
int starpu_mpi_checkpoint_template_create (
            starpu_mpi_checkpoint_template_t * cp_template,
            int cp_id,
            int cp_domain )
```

Create a new checkpoint template. A unique checkpoint id `cp_id` is requested from the user in order to create several templates and to match with a corresponding ::starpu_mpi_init_from_checkpoint() (not implemented yet). Note a template must be frozen with starpu_mpi_checkpoint_template_freeze() in order to use it with starpu_mpi_checkpoint_template_submit().

57.21.2.5 starpu_mpi_checkpoint_template_add_entry()

```
int starpu_mpi_checkpoint_template_add_entry (
            starpu_mpi_checkpoint_template_t * cp_template,
             ... )
```

Add a single entry to a checkpoint template previously created with starpu_mpi_checkpoint_template_create(). As many entries can be added to a template with as many argument to a single function call, or with as many calls to this function. Once all the entry added, the template must be frozen before using starpu_mpi_checkpoint_template_submit().

The arguments following the `cp_template` can be of the following types:

- STARPU_R followed by a data handle and the backup rank;
- (non functional) STARPU_DATA_ARRAY followed by an array of data handles, its number of elements and a backup rank (non functional);
- STARPU_VALUE followed by a pointer to the unregistered value, its size in bytes, a unique tag (as the ones given for data handle registering) and the function giving the back up rank of the rank argument : int(backup↩_of)(int) .
- The argument list must be ended by the value 0.

57.21.2.6 starpu_mpi_checkpoint_template_freeze()

```
int starpu_mpi_checkpoint_template_freeze (
            starpu_mpi_checkpoint_template_t * cp_template )
```

Freeze the given template. A frozen template can no longer be modified with starpu_mpi_checkpoint_template_add_entry(). A template must be frozen before using starpu_mpi_checkpoint_template_submit().

57.21.2.7 starpu_mpi_checkpoint_template_submit()

```
int starpu_mpi_checkpoint_template_submit (
            starpu_mpi_checkpoint_template_t cp_template,
            int prio )
```

Submit the checkpoint to StarPU, and can be seen as a cut in the task graph. StarPU will save the data as currently described in the submission. Note that the data external to StarPu (STARPU_VALUE) will be saved with the current value at submission time (when starpu_mpi_checkpoint_template_submit() is called). The data internal to StarPU (aka handles given with STARPU_R) will be saved with their value at execution time (when the task submitted before the starpu_mpi_checkpoint_template_submit() have been executed, and before this data is modified by the tasks submitted after the starpu_mpi_checkpoint_template_submit())

57.22 MPI Support

Data Structures

- struct starpu_mpi_task_exchange_params

Macros

- #define STARPU_USE_MPI_MASTER_SLAVE
- #define STARPU_USE_MPI
- #define STARPU_FXT_MAX_FILES
- #define STARPU_EXECUTE_ON_NODE
- #define STARPU_EXECUTE_ON_DATA
- #define STARPU_NODE_SELECTION_POLICY

Variables

- int starpu_mpi_task_exchange_params::do_execute
- int starpu_mpi_task_exchange_params::xrank
- int starpu_mpi_task_exchange_params::priority

Communication

- typedef void ∗ starpu_mpi_req
- typedef int64_t starpu_mpi_tag_t
- typedef int(∗ **starpu_mpi_datatype_allocate_func_t**) (starpu_data_handle_t, MPI_Datatype ∗)
- typedef int(∗ **starpu_mpi_datatype_node_allocate_func_t**) (starpu_data_handle_t, unsigned node, MPI_Datatype ∗)
- typedef void(∗ **starpu_mpi_datatype_free_func_t**) (MPI_Datatype ∗)
- int starpu_mpi_isend (starpu_data_handle_t data_handle, starpu_mpi_req ∗req, int dest, starpu_mpi_tag_t data_tag, MPI_Comm comm)
- int starpu_mpi_isend_prio (starpu_data_handle_t data_handle, starpu_mpi_req ∗req, int dest, starpu_mpi_tag_t data_tag, int prio, MPI_Comm comm)
- int starpu_mpi_irecv (starpu_data_handle_t data_handle, starpu_mpi_req ∗req, int source, starpu_mpi_tag_t data_tag, MPI_Comm comm)
- int starpu_mpi_send (starpu_data_handle_t data_handle, int dest, starpu_mpi_tag_t data_tag, MPI_Comm comm)
- int starpu_mpi_send_prio (starpu_data_handle_t data_handle, int dest, starpu_mpi_tag_t data_tag, int prio, MPI_Comm comm)
- int starpu_mpi_recv (starpu_data_handle_t data_handle, int source, starpu_mpi_tag_t data_tag, MPI_Comm comm, MPI_Status ∗status)
- int starpu_mpi_recv_prio (starpu_data_handle_t data_handle, int source, starpu_mpi_tag_t data_tag, int prio, MPI_Comm comm, MPI_Status ∗status)
- int starpu_mpi_isend_detached (starpu_data_handle_t data_handle, int dest, starpu_mpi_tag_t data_tag, MPI_Comm comm, void(∗callback)(void ∗), void ∗arg)
- int starpu_mpi_isend_detached_prio (starpu_data_handle_t data_handle, int dest, starpu_mpi_tag_t data_tag, int prio, MPI_Comm comm, void(∗callback)(void ∗), void ∗arg)
- int starpu_mpi_irecv_detached (starpu_data_handle_t data_handle, int source, starpu_mpi_tag_t data_tag, MPI_Comm comm, void(∗callback)(void ∗), void ∗arg)
- int starpu_mpi_irecv_detached_prio (starpu_data_handle_t data_handle, int source, starpu_mpi_tag_t data_tag, int prio, MPI_Comm comm, void(∗callback)(void ∗), void ∗arg)
- int starpu_mpi_irecv_detached_sequential_consistency (starpu_data_handle_t data_handle, int source, starpu_mpi_tag_t data_tag, MPI_Comm comm, void(∗callback)(void ∗), void ∗arg, int sequential_consistency)
- int starpu_mpi_issend (starpu_data_handle_t data_handle, starpu_mpi_req ∗req, int dest, starpu_mpi_tag_t data_tag, MPI_Comm comm)

- int starpu_mpi_issend_prio (starpu_data_handle_t data_handle, starpu_mpi_req ∗req, int dest, starpu_mpi_tag_t data_tag, int prio, MPI_Comm comm)
- int starpu_mpi_issend_detached (starpu_data_handle_t data_handle, int dest, starpu_mpi_tag_t data_tag, MPI_Comm comm, void(∗callback)(void ∗), void ∗arg)
- int starpu_mpi_issend_detached_prio (starpu_data_handle_t data_handle, int dest, starpu_mpi_tag_t data↩ _tag, int prio, MPI_Comm comm, void(∗callback)(void ∗), void ∗arg)
- int starpu_mpi_wait (starpu_mpi_req ∗req, MPI_Status ∗status)
- int starpu_mpi_test (starpu_mpi_req ∗req, int ∗flag, MPI_Status ∗status)
- int starpu_mpi_barrier (MPI_Comm comm)
- int starpu_mpi_wait_for_all (MPI_Comm comm)
- int starpu_mpi_isend_detached_unlock_tag (starpu_data_handle_t data_handle, int dest, starpu_mpi_tag_t data_tag, MPI_Comm comm, starpu_tag_t tag)
- int starpu_mpi_isend_detached_unlock_tag_prio (starpu_data_handle_t data_handle, int dest, starpu_mpi_tag_t data_tag, int prio, MPI_Comm comm, starpu_tag_t tag)
- int starpu_mpi_irecv_detached_unlock_tag (starpu_data_handle_t data_handle, int source, starpu_mpi_tag_t data_tag, MPI_Comm comm, starpu_tag_t tag)
- int starpu_mpi_isend_array_detached_unlock_tag (unsigned array_size, starpu_data_handle_t ∗data_↩ handle, int ∗dest, starpu_mpi_tag_t ∗data_tag, MPI_Comm ∗comm, starpu_tag_t tag)
- int starpu_mpi_isend_array_detached_unlock_tag_prio (unsigned array_size, starpu_data_handle_t ∗data↩ _handle, int ∗dest, starpu_mpi_tag_t ∗data_tag, int ∗prio, MPI_Comm ∗comm, starpu_tag_t tag)
- int starpu_mpi_irecv_array_detached_unlock_tag (unsigned array_size, starpu_data_handle_t ∗data_↩ handle, int ∗source, starpu_mpi_tag_t ∗data_tag, MPI_Comm ∗comm, starpu_tag_t tag)
- int starpu_mpi_datatype_register (starpu_data_handle_t handle, starpu_mpi_datatype_allocate_func_↩ t allocate_datatype_func, starpu_mpi_datatype_free_func_t free_datatype_func)
- int starpu_mpi_interface_datatype_register (enum starpu_data_interface_id id, starpu_mpi_datatype_↩ allocate_func_t allocate_datatype_func, starpu_mpi_datatype_free_func_t free_datatype_func)
- int starpu_mpi_datatype_node_register (starpu_data_handle_t handle, starpu_mpi_datatype_node_↩ allocate_func_t allocate_datatype_func, starpu_mpi_datatype_free_func_t free_datatype_func)
- int starpu_mpi_interface_datatype_node_register (enum starpu_data_interface_id id, starpu_mpi_datatype↩ _node_allocate_func_t allocate_datatype_func, starpu_mpi_datatype_free_func_t free_datatype_func)
- int starpu_mpi_datatype_unregister (starpu_data_handle_t handle)
- int starpu_mpi_interface_datatype_unregister (enum starpu_data_interface_id id)

Node Selection Policy

- typedef int(∗ **starpu_mpi_select_node_policy_func_t**) (int me, int nb_nodes, struct starpu_data_descr ∗descr, int nb_data)
- int starpu_mpi_node_selection_register_policy (starpu_mpi_select_node_policy_func_t policy_func)
- int starpu_mpi_node_selection_unregister_policy (int policy)
- int starpu_mpi_node_selection_get_current_policy (void)
- int starpu_mpi_node_selection_set_current_policy (int policy)
- #define STARPU_MPI_NODE_SELECTION_CURRENT_POLICY
- #define STARPU_MPI_NODE_SELECTION_MOST_R_DATA

Initialisation

- int starpu_mpi_init_conf (int ∗argc, char ∗∗∗argv, int initialize_mpi, MPI_Comm comm, struct starpu_conf ∗conf)
- int starpu_mpi_init_comm (int ∗argc, char ∗∗∗argv, int initialize_mpi, MPI_Comm comm)
- int starpu_mpi_init (int ∗argc, char ∗∗∗argv, int initialize_mpi)
- int starpu_mpi_initialize (void)
- int starpu_mpi_initialize_extended (int ∗rank, int ∗world_size)
- int starpu_mpi_shutdown (void)
- int starpu_mpi_shutdown_comm (MPI_Comm comm)
- int starpu_mpi_comm_register (MPI_Comm comm)
- int starpu_mpi_comm_size (MPI_Comm comm, int ∗size)

- int starpu_mpi_comm_rank (MPI_Comm comm, int ∗rank)
- int starpu_mpi_world_rank (void)
- int starpu_mpi_world_size (void)
- int starpu_mpi_comm_get_attr (MPI_Comm comm, int keyval, void ∗attribute_val, int ∗flag)
- int starpu_mpi_get_thread_cpuid (void)
- int starpu_mpi_get_communication_tag (void)
- void starpu_mpi_set_communication_tag (int tag)
- #define STARPU_MPI_TAG_UB

Communication Cache

- int starpu_mpi_cache_is_enabled (void)
- int starpu_mpi_cache_set (int enabled)
- void starpu_mpi_cache_flush (MPI_Comm comm, starpu_data_handle_t data_handle)
- void starpu_mpi_cache_flush_all_data (MPI_Comm comm)
- int starpu_mpi_cached_receive (starpu_data_handle_t data_handle)
- int starpu_mpi_cached_receive_set (starpu_data_handle_t data)
- int **starpu_mpi_cached_cp_receive_set** (starpu_data_handle_t data_handle)
- void starpu_mpi_cached_receive_clear (starpu_data_handle_t data)
- int starpu_mpi_cached_send (starpu_data_handle_t data_handle, int dest)
- int starpu_mpi_cached_send_set (starpu_data_handle_t data, int dest)
- void starpu_mpi_cached_send_clear (starpu_data_handle_t data)

MPI Insert Task

- void starpu_mpi_data_register_comm (starpu_data_handle_t data_handle, starpu_mpi_tag_t data_tag, int rank, MPI_Comm comm)
- void starpu_mpi_data_set_tag (starpu_data_handle_t handle, starpu_mpi_tag_t data_tag)
- void starpu_mpi_data_set_rank_comm (starpu_data_handle_t handle, int rank, MPI_Comm comm)
- int starpu_mpi_data_get_rank (starpu_data_handle_t handle)
- starpu_mpi_tag_t starpu_mpi_data_get_tag (starpu_data_handle_t handle)
- char ∗ starpu_mpi_data_get_redux_map (starpu_data_handle_t handle)
- int starpu_mpi_task_insert (MPI_Comm comm, struct starpu_codelet ∗codelet,...)
- int starpu_mpi_insert_task (MPI_Comm comm, struct starpu_codelet ∗codelet,...)
- struct starpu_task ∗ starpu_mpi_task_build (MPI_Comm comm, struct starpu_codelet ∗codelet,...)
- struct starpu_task ∗ starpu_mpi_task_build_v (MPI_Comm comm, struct starpu_codelet ∗codelet, va_list varg_list)
- int starpu_mpi_task_post_build (MPI_Comm comm, struct starpu_codelet ∗codelet,...)
- int starpu_mpi_task_post_build_v (MPI_Comm comm, struct starpu_codelet ∗codelet, va_list varg_list)
- int starpu_mpi_task_exchange_data_before_execution (MPI_Comm comm, struct starpu_task ∗task, struct starpu_data_descr ∗descrs, struct starpu_mpi_task_exchange_params ∗params)
- int starpu_mpi_task_exchange_data_after_execution (MPI_Comm comm, struct starpu_data_descr ∗descrs, unsigned nb_data, struct starpu_mpi_task_exchange_params params)
- int starpu_mpi_get_data_on_node (MPI_Comm comm, starpu_data_handle_t data_handle, int node)
- int starpu_mpi_get_data_on_node_detached (MPI_Comm comm, starpu_data_handle_t data_handle, int node, void(∗callback)(void ∗), void ∗arg)
- void starpu_mpi_get_data_on_all_nodes_detached (MPI_Comm comm, starpu_data_handle_t data_handle)
- void starpu_mpi_data_migrate (MPI_Comm comm, starpu_data_handle_t handle, int new_rank)
- #define STARPU_MPI_PER_NODE
- #define starpu_mpi_data_register(data_handle, data_tag, rank)
- #define starpu_data_set_tag
- #define starpu_mpi_data_set_rank(handle, rank)
- #define starpu_data_set_rank
- #define starpu_data_get_rank
- #define starpu_data_get_tag

Collective Operations

- int starpu_mpi_redux_data (MPI_Comm comm, starpu_data_handle_t data_handle)
- int starpu_mpi_redux_data_prio (MPI_Comm comm, starpu_data_handle_t data_handle, int prio)
- int starpu_mpi_redux_data_tree (MPI_Comm comm, starpu_data_handle_t data_handle, int arity)
- int starpu_mpi_redux_data_prio_tree (MPI_Comm comm, starpu_data_handle_t data_handle, int prio, int arity)
- int starpu_mpi_scatter_detached (starpu_data_handle_t ∗data_handles, int count, int root, MPI_Comm comm, void(∗scallback)(void ∗), void ∗sarg, void(∗rcallback)(void ∗), void ∗rarg)
- int starpu_mpi_gather_detached (starpu_data_handle_t ∗data_handles, int count, int root, MPI_Comm comm, void(∗scallback)(void ∗), void ∗sarg, void(∗rcallback)(void ∗), void ∗rarg)

Dynamic Broadcasts

- void starpu_mpi_coop_sends_set_use (int use_coop_sends)
- int starpu_mpi_coop_sends_get_use (void)
- void starpu_mpi_coop_sends_data_handle_nb_sends (starpu_data_handle_t data_handle, int nb_sends)

Statistics

- void starpu_mpi_comm_stats_disable (void)
- void starpu_mpi_comm_stats_enable (void)
- void starpu_mpi_comm_stats_retrieve (size_t ∗comm_stats)

Miscellaneous

- int **starpu_mpi_pre_submit_hook_register** (void(∗f)(struct starpu_task ∗))
- int **starpu_mpi_pre_submit_hook_unregister** (void)
- int starpu_mpi_data_cpy (starpu_data_handle_t dst_handle, starpu_data_handle_t src_handle, MPI_Comm comm, int asynchronous, void(∗callback_func)(void ∗), void ∗callback_arg)
- int starpu_mpi_data_cpy_priority (starpu_data_handle_t dst_handle, starpu_data_handle_t src_handle, MPI_Comm comm, int asynchronous, void(∗callback_func)(void ∗), void ∗callback_arg, int priority)

Data Tags Management

- int64_t starpu_mpi_tags_allocate (int64_t nbtags)
- void starpu_mpi_tags_free (int64_t mintag)

57.22.1 Detailed Description

57.22.2 Data Structure Documentation

57.22.2.1 struct starpu_mpi_task_exchange_params

Structure used to pass data from starpu_mpi_task_exchange_data_before_execution() to starpu_mpi_task_exchange_data_after_exe

Data Fields

int	do_execute	is the caller going to execute the task
int	xrank	node executing the task
int	priority	priority of the task being executed

57.22.3 Macro Definition Documentation

57.22.3.1 STARPU_USE_MPI_MASTER_SLAVE

`#define STARPU_USE_MPI_MASTER_SLAVE`

Defined when StarPU has been installed with MPI Master Slave support. It should be used in your code to detect the availability of MPI Master Slave.

57.22.3.2 STARPU_USE_MPI

`#define STARPU_USE_MPI`

Defined when StarPU has been installed with MPI support. It should be used in your code to detect the availability of MPI.

57.22.3.3 STARPU_FXT_MAX_FILES

`#define STARPU_FXT_MAX_FILES`

Define the maximum number of fxt mpi files that can be read when generating traces. The default value is 64, it can be changed by using the configure option --enable-fxt-max-files.

57.22.3.4 STARPU_EXECUTE_ON_NODE

`#define STARPU_EXECUTE_ON_NODE`

Used when calling starpu_mpi_task_insert(), must be followed by a integer value which specified the node on which to execute the codelet.

57.22.3.5 STARPU_EXECUTE_ON_DATA

`#define STARPU_EXECUTE_ON_DATA`

Used when calling starpu_mpi_task_insert(), must be followed by a data handle to specify that the node owning the given data will execute the codelet.

57.22.3.6 STARPU_NODE_SELECTION_POLICY

`#define STARPU_NODE_SELECTION_POLICY`

Used when calling starpu_mpi_task_insert(), must be followed by a identifier to a node selection policy. This is needed when several nodes own data in STARPU_W mode.

57.22.3.7 STARPU_MPI_TAG_UB

`#define STARPU_MPI_TAG_UB`

When given to the function starpu_mpi_comm_get_attr(), retrieve the value for the upper bound for tag value.

57.22.3.8 STARPU_MPI_PER_NODE

`#define STARPU_MPI_PER_NODE`

Can be used as rank when calling starpu_mpi_data_register() and alike, to specify that the data is per-node: each node will have its own value. Tasks writing to such data will be replicated on all nodes (and all parameters then have to be per-node). Tasks not writing to such data will just take the node-local value without any MPI communication.

57.22.3.9 starpu_mpi_data_register

```
#define starpu_mpi_data_register(
            data_handle,
            data_tag,
            rank )
```

Register to MPI a StarPU data handle with the given tag, rank and the MPI communicator `MPI_COMM_WORLD`. It also automatically clears the MPI communication cache when unregistering the data.

57.22.3.10 starpu_data_set_tag

`#define starpu_data_set_tag`

Symbol kept for backward compatibility. Call function starpu_mpi_data_set_tag()

57.22.3.11 starpu_mpi_data_set_rank

```
#define starpu_mpi_data_set_rank(
              handle,
              rank )
```

Register to MPI a StarPU data handle with the given rank and the MPI communicator `MPI_COMM_WORLD`. No tag will be defined. It also automatically clears the MPI communication cache when unregistering the data.

57.22.3.12 starpu_data_set_rank

```
#define starpu_data_set_rank
```

Symbol kept for backward compatibility. Call function starpu_mpi_data_set_rank()

57.22.3.13 starpu_data_get_rank

```
#define starpu_data_get_rank
```

Symbol kept for backward compatibility. Call function starpu_mpi_data_get_rank()

57.22.3.14 starpu_data_get_tag

```
#define starpu_data_get_tag
```

Symbol kept for backward compatibility. Call function starpu_mpi_data_get_tag()

57.22.3.15 STARPU_MPI_NODE_SELECTION_CURRENT_POLICY

```
#define STARPU_MPI_NODE_SELECTION_CURRENT_POLICY
```

Define the current policy

57.22.3.16 STARPU_MPI_NODE_SELECTION_MOST_R_DATA

```
#define STARPU_MPI_NODE_SELECTION_MOST_R_DATA
```

Define the policy in which the selected node is the one having the most data in STARPU_R mode

57.22.4 Typedef Documentation

57.22.4.1 starpu_mpi_req

```
typedef void* starpu_mpi_req
```

Opaque type for communication request

57.22.4.2 starpu_mpi_tag_t

```
typedef int64_t starpu_mpi_tag_t
```

Type of the message tag.

57.22.5 Function Documentation

57.22.5.1 starpu_mpi_init_conf()

```
int starpu_mpi_init_conf (
            int * argc,
            char *** argv,
            int initialize_mpi,
            MPI_Comm comm,
            struct starpu_conf * conf )
```

Initialize the StarPU library with the given `conf`, and initialize the StarPU-MPI library with the given MPI communicator `comm`. `initialize_mpi` indicates if MPI should be initialized or not by StarPU. StarPU-MPI takes the opportunity to modify `conf` to either reserve a core for its MPI thread (by default), or execute MPI calls on the CPU driver 0 between tasks.

57.22.5.2 starpu_mpi_init_comm()

```
int starpu_mpi_init_comm (
            int * argc,
            char *** argv,
            int initialize_mpi,
            MPI_Comm comm )
```

Same as starpu_mpi_init_conf(), except that this does not initialize the StarPU library. The caller thus has to call starpu_init() before this, and it can not reserve a core for the MPI communications.

57.22.5.3 starpu_mpi_init()

```
int starpu_mpi_init (
            int * argc,
            char *** argv,
            int initialize_mpi )
```

Call starpu_mpi_init_comm() with the MPI communicator `MPI_COMM_WORLD`.

57.22.5.4 starpu_mpi_initialize()

```
int starpu_mpi_initialize (
            void )
```

Deprecated This function has been made deprecated. One should use instead the function starpu_mpi_init(). This function does not call `MPI_Init()`, it should be called beforehand.

57.22.5.5 starpu_mpi_initialize_extended()

```
int starpu_mpi_initialize_extended (
            int * rank,
            int * world_size )
```

Deprecated This function has been made deprecated. One should use instead the function starpu_mpi_init(). MPI will be initialized by starpumpi by calling `MPI_Init_Thread(argc, argv, MPI_THREAD_SERIALIZED, ...)`.

57.22.5.6 starpu_mpi_shutdown()

```
int starpu_mpi_shutdown (
            void )
```

Call starpu_mpi_shutdown_comm() with the MPI communicator `MPI_COMM_WORLD`

57.22.5.7 starpu_mpi_shutdown_comm()

```
int starpu_mpi_shutdown_comm (
            MPI_Comm comm )
```

Clean the starpumpi library. This must be called after calling any `starpu_mpi` functions and before the call to starpu_shutdown(), if any. `MPI_Finalize()` will be called if StarPU-MPI has been initialized by starpu_mpi_init().

57.22.5.8 starpu_mpi_comm_register()

```
int starpu_mpi_comm_register (
            MPI_Comm comm )
```

Register `comm`. The function is automatically called for the communicator given to starpu_mpi_init_comm().

57.22.5.9 starpu_mpi_comm_size()

```
int starpu_mpi_comm_size (
            MPI_Comm comm,
            int * size )
```

Return in `size` the size of the communicator `comm`. The function will fail if starpu_mpi_comm_register() has not been previously called with the given communicator.

57.22.5.10 starpu_mpi_comm_rank()

```
int starpu_mpi_comm_rank (
            MPI_Comm comm,
            int * rank )
```

Return in `rank` the rank of the calling process in the communicator `comm`. The function will fail if starpu_mpi_comm_register() has not been previously called with the given communicator.

57.22.5.11 starpu_mpi_world_rank()

```
int starpu_mpi_world_rank (
            void )
```

Return the rank of the calling process in the communicator `MPI_COMM_WORLD`

57.22.5.12 starpu_mpi_world_size()

```
int starpu_mpi_world_size (
            void )
```

Return the size of the communicator `MPI_COMM_WORLD`

57.22.5.13 starpu_mpi_comm_get_attr()

```
int starpu_mpi_comm_get_attr (
            MPI_Comm comm,
            int keyval,
            void * attribute_val,
            int * flag )
```

Retrieve an attribute value by key, similarly to the MPI function `MPI_comm_get_attr()`, except that the value is a pointer to int64_t instead of int. If an attribute is attached on `comm` to `keyval`, then the call returns `flag` equal to `1`, and the attribute value in `attribute_val`. Otherwise, `flag` is set to \0.

57.22.5.14 starpu_mpi_get_thread_cpuid()

```
int starpu_mpi_get_thread_cpuid (
            void )
```

Get the logical index of the core where the MPI thread is bound.

57.22.5.15 starpu_mpi_get_communication_tag()

```
int starpu_mpi_get_communication_tag (
            void )
```

Get the tag used for MPI communications submitted by StarPU.

57.22.5.16 starpu_mpi_set_communication_tag()

```
void starpu_mpi_set_communication_tag (
             int tag )
```

Set the tag used for MPI communications submitted by StarPU.

57.22.5.17 starpu_mpi_isend()

```
int starpu_mpi_isend (
             starpu_data_handle_t data_handle,
             starpu_mpi_req * req,
             int dest,
             starpu_mpi_tag_t data_tag,
             MPI_Comm comm )
```

Post a standard-mode, non blocking send of `data_handle` to the node `dest` using the message tag `data_tag` within the communicator `comm`. After the call, the pointer to the request `req` can be used to test or to wait for the completion of the communication.

57.22.5.18 starpu_mpi_isend_prio()

```
int starpu_mpi_isend_prio (
             starpu_data_handle_t data_handle,
             starpu_mpi_req * req,
             int dest,
             starpu_mpi_tag_t data_tag,
             int prio,
             MPI_Comm comm )
```

Similar to starpu_mpi_isend(), but take a priority `prio`.

57.22.5.19 starpu_mpi_irecv()

```
int starpu_mpi_irecv (
             starpu_data_handle_t data_handle,
             starpu_mpi_req * req,
             int source,
             starpu_mpi_tag_t data_tag,
             MPI_Comm comm )
```

Post a nonblocking receive in `data_handle` from the node `source` using the message tag `data_tag` within the communicator `comm`. After the call, the pointer to the request `req` can be used to test or to wait for the completion of the communication.

57.22.5.20 starpu_mpi_send()

```
int starpu_mpi_send (
             starpu_data_handle_t data_handle,
             int dest,
             starpu_mpi_tag_t data_tag,
             MPI_Comm comm )
```

Perform a standard-mode, blocking send of `data_handle` to the node `dest` using the message tag `data_tag` within the communicator `comm`.

57.22.5.21 starpu_mpi_send_prio()

```
int starpu_mpi_send_prio (
             starpu_data_handle_t data_handle,
             int dest,
             starpu_mpi_tag_t data_tag,
             int prio,
             MPI_Comm comm )
```

Similar to starpu_mpi_send(), but take a priority `prio`.

57.22.5.22 starpu_mpi_recv()

```
int starpu_mpi_recv (
            starpu_data_handle_t data_handle,
            int source,
            starpu_mpi_tag_t data_tag,
            MPI_Comm comm,
            MPI_Status * status )
```

Perform a standard-mode, blocking receive in `data_handle` from the node `source` using the message tag `data_tag` within the communicator `comm`. The value of `status` cannot be NULL, use the predefined value MPI_STATUS_IGNORE to ignore the status.

57.22.5.23 starpu_mpi_recv_prio()

```
int starpu_mpi_recv_prio (
            starpu_data_handle_t data_handle,
            int source,
            starpu_mpi_tag_t data_tag,
            int prio,
            MPI_Comm comm,
            MPI_Status * status )
```

Similar to starpu_mpi_recv(), but take a priority `prio`

57.22.5.24 starpu_mpi_isend_detached()

```
int starpu_mpi_isend_detached (
            starpu_data_handle_t data_handle,
            int dest,
            starpu_mpi_tag_t data_tag,
            MPI_Comm comm,
            void(*)(void *) callback,
            void * arg )
```

Post a standard-mode, non blocking send of `data_handle` to the node `dest` using the message tag `data_tag` within the communicator `comm`. On completion, the `callback` function is called with the argument `arg`. Similarly to the pthread detached functionality, when a detached communication completes, its resources are automatically released back to the system, there is no need to test or to wait for the completion of the request.

57.22.5.25 starpu_mpi_isend_detached_prio()

```
int starpu_mpi_isend_detached_prio (
            starpu_data_handle_t data_handle,
            int dest,
            starpu_mpi_tag_t data_tag,
            int prio,
            MPI_Comm comm,
            void(*)(void *) callback,
            void * arg )
```

Similar to starpu_mpi_isend_detached(), but take a priority `prio`.

57.22.5.26 starpu_mpi_irecv_detached()

```
int starpu_mpi_irecv_detached (
            starpu_data_handle_t data_handle,
            int source,
            starpu_mpi_tag_t data_tag,
            MPI_Comm comm,
            void(*)(void *) callback,
            void * arg )
```

Post a nonblocking receive in `data_handle` from the node `source` using the message tag `data_tag` within the communicator `comm`. On completion, the `callback` function is called with the argument `arg`. Similarly to the pthread detached functionality, when a detached communication completes, its resources are automatically released back to the system, there is no need to test or to wait for the completion of the request.

57.22.5.27 starpu_mpi_irecv_detached_prio()

```
int starpu_mpi_irecv_detached_prio (
            starpu_data_handle_t data_handle,
            int source,
            starpu_mpi_tag_t data_tag,
            int prio,
            MPI_Comm comm,
            void(*)(void *) callback,
            void * arg )
```

Same of starpu_mpi_irecv_detached but with the `prio` parameter.

57.22.5.28 starpu_mpi_irecv_detached_sequential_consistency()

```
int starpu_mpi_irecv_detached_sequential_consistency (
            starpu_data_handle_t data_handle,
            int source,
            starpu_mpi_tag_t data_tag,
            MPI_Comm comm,
            void(*)(void *) callback,
            void * arg,
            int sequential_consistency )
```

Post a nonblocking receive in `data_handle` from the node `source` using the message tag `data_tag` within the communicator `comm`. On completion, the `callback` function is called with the argument `arg`. The parameter `sequential_consistency` allows to enable or disable the sequential consistency for `data` handle (sequential consistency will be enabled or disabled based on the value of the parameter `sequential_consistency` and the value of the sequential consistency defined for `data_handle`). Similarly to the pthread detached functionality, when a detached communication completes, its resources are automatically released back to the system, there is no need to test or to wait for the completion of the request.

57.22.5.29 starpu_mpi_issend()

```
int starpu_mpi_issend (
            starpu_data_handle_t data_handle,
            starpu_mpi_req * req,
            int dest,
            starpu_mpi_tag_t data_tag,
            MPI_Comm comm )
```

Perform a synchronous-mode, non-blocking send of `data_handle` to the node `dest` using the message tag `data_tag` within the communicator `comm`.

57.22.5.30 starpu_mpi_issend_prio()

```
int starpu_mpi_issend_prio (
            starpu_data_handle_t data_handle,
            starpu_mpi_req * req,
            int dest,
            starpu_mpi_tag_t data_tag,
            int prio,
            MPI_Comm comm )
```

Similar to starpu_mpi_issend(), but take a priority `prio`.

57.22.5.31 starpu_mpi_issend_detached()

```
int starpu_mpi_issend_detached (
            starpu_data_handle_t data_handle,
            int dest,
            starpu_mpi_tag_t data_tag,
            MPI_Comm comm,
            void(*)(void *) callback,
            void * arg )
```

Perform a synchronous-mode, non-blocking send of `data_handle` to the node `dest` using the message tag `data_tag` within the communicator `comm`. On completion, the `callback` function is called with the argument `arg`. Similarly to the pthread detached functionality, when a detached communication completes, its resources are automatically released back to the system, there is no need to test or to wait for the completion of the request.

57.22.5.32 starpu_mpi_issend_detached_prio()

```
int starpu_mpi_issend_detached_prio (
            starpu_data_handle_t data_handle,
            int dest,
            starpu_mpi_tag_t data_tag,
            int prio,
            MPI_Comm comm,
            void(*)(void *) callback,
            void * arg )
```

Similar to starpu_mpi_issend_detached(), but take a priority `prio`.

57.22.5.33 starpu_mpi_wait()

```
int starpu_mpi_wait (
            starpu_mpi_req * req,
            MPI_Status * status )
```

Return when the operation identified by request `req` is complete. The value of `status` cannot be NULL, use the predefined value MPI_STATUS_IGNORE to ignore the status.

57.22.5.34 starpu_mpi_test()

```
int starpu_mpi_test (
            starpu_mpi_req * req,
            int * flag,
            MPI_Status * status )
```

If the operation identified by `req` is complete, set `flag` to

1. The `status` object is set to contain information on the completed operation.

57.22.5.35 starpu_mpi_barrier()

```
int starpu_mpi_barrier (
            MPI_Comm comm )
```

Block the caller until all group members of the communicator `comm` have called it.

57.22.5.36 starpu_mpi_wait_for_all()

```
int starpu_mpi_wait_for_all (
            MPI_Comm comm )
```

Wait until all StarPU tasks and communications for the given communicator are completed.

57.22.5.37 starpu_mpi_isend_detached_unlock_tag()

```
int starpu_mpi_isend_detached_unlock_tag (
            starpu_data_handle_t data_handle,
            int dest,
            starpu_mpi_tag_t data_tag,
            MPI_Comm comm,
            starpu_tag_t tag )
```

Post a standard-mode, non blocking send of `data_handle` to the node `dest` using the message tag `data_tag` within the communicator `comm`. On completion, `tag` is unlocked.

57.22.5.38 starpu_mpi_isend_detached_unlock_tag_prio()

```
int starpu_mpi_isend_detached_unlock_tag_prio (
            starpu_data_handle_t data_handle,
            int dest,
            starpu_mpi_tag_t data_tag,
            int prio,
            MPI_Comm comm,
            starpu_tag_t tag )
```

Similar to starpu_mpi_isend_detached_unlock_tag(), but take a priority `prio`.

57.22.5.39 starpu_mpi_irecv_detached_unlock_tag()

```
int starpu_mpi_irecv_detached_unlock_tag (
            starpu_data_handle_t data_handle,
            int source,
            starpu_mpi_tag_t data_tag,
            MPI_Comm comm,
            starpu_tag_t tag )
```

Post a nonblocking receive in `data_handle` from the node `source` using the message tag `data_tag` within the communicator `comm`. On completion, `tag` is unlocked.

57.22.5.40 starpu_mpi_isend_array_detached_unlock_tag()

```
int starpu_mpi_isend_array_detached_unlock_tag (
            unsigned array_size,
            starpu_data_handle_t * data_handle,
            int * dest,
            starpu_mpi_tag_t * data_tag,
            MPI_Comm * comm,
            starpu_tag_t tag )
```

Post `array_size` standard-mode, non blocking send. Each post sends the n-th data of the array `data_handle` to the n-th node of the array `dest` using the n-th message tag of the array `data_tag` within the n-th communicator of the array `comm`. On completion of the all the requests, `tag` is unlocked.

57.22.5.41 starpu_mpi_isend_array_detached_unlock_tag_prio()

```
int starpu_mpi_isend_array_detached_unlock_tag_prio (
            unsigned array_size,
            starpu_data_handle_t * data_handle,
            int * dest,
            starpu_mpi_tag_t * data_tag,
            int * prio,
            MPI_Comm * comm,
            starpu_tag_t tag )
```

Similar to starpu_mpi_isend_array_detached_unlock_tag(), but take a priority `prio`.

57.22.5.42 starpu_mpi_irecv_array_detached_unlock_tag()

```
int starpu_mpi_irecv_array_detached_unlock_tag (
            unsigned array_size,
            starpu_data_handle_t * data_handle,
            int * source,
            starpu_mpi_tag_t * data_tag,
            MPI_Comm * comm,
            starpu_tag_t tag )
```

Post `array_size` nonblocking receive. Each post receives in the n-th data of the array `data_handle` from the n-th node of the array `source` using the n-th message tag of the array `data_tag` within the n-th communicator of the array `comm`. On completion of the all the requests, `tag` is unlocked.

57.22.5.43 starpu_mpi_datatype_register()

```
int starpu_mpi_datatype_register (
            starpu_data_handle_t handle,
            starpu_mpi_datatype_allocate_func_t allocate_datatype_func,
            starpu_mpi_datatype_free_func_t free_datatype_func )
```

Register functions to create and free a MPI datatype for the given handle. Similar to starpu_mpi_interface_datatype_register(). It is important that the function is called before any communication can take place for a data with the given handle. See Exchanging User Defined Data Interface for an example.

57.22.5.44 starpu_mpi_interface_datatype_register()

```
int starpu_mpi_interface_datatype_register (
            enum starpu_data_interface_id id,
            starpu_mpi_datatype_allocate_func_t allocate_datatype_func,
            starpu_mpi_datatype_free_func_t free_datatype_func )
```

Register functions to create and free a MPI datatype for the given interface id. Similar to starpu_mpi_datatype_register(). It is important that the function is called before any communication can take place for a data with the given handle. See Exchanging User Defined Data Interface for an example.

57.22.5.45 starpu_mpi_datatype_node_register()

```
int starpu_mpi_datatype_node_register (
            starpu_data_handle_t handle,
            starpu_mpi_datatype_node_allocate_func_t allocate_datatype_func,
            starpu_mpi_datatype_free_func_t free_datatype_func )
```

Register functions to create and free a MPI datatype for the given handle. Similar to starpu_mpi_interface_datatype_register(). It is important that the function is called before any communication can take place for a data with the given handle. See Exchanging User Defined Data Interface for an example.

57.22.5.46 starpu_mpi_interface_datatype_node_register()

```
int starpu_mpi_interface_datatype_node_register (
            enum starpu_data_interface_id id,
            starpu_mpi_datatype_node_allocate_func_t allocate_datatype_func,
            starpu_mpi_datatype_free_func_t free_datatype_func )
```

Register functions to create and free a MPI datatype for the given interface id. Similar to starpu_mpi_datatype_register(). It is important that the function is called before any communication can take place for a data with the given handle. See Exchanging User Defined Data Interface for an example.

57.22.5.47 starpu_mpi_datatype_unregister()

```
int starpu_mpi_datatype_unregister (
            starpu_data_handle_t handle )
```

Unregister the MPI datatype functions stored for the interface of the given handle.

57.22.5.48 starpu_mpi_interface_datatype_unregister()

```
int starpu_mpi_interface_datatype_unregister (
            enum starpu_data_interface_id id )
```

Unregister the MPI datatype functions stored for the interface of the given interface id. Similar to starpu_mpi_datatype_unregister().

57.22.5.49 starpu_mpi_cache_is_enabled()

```
int starpu_mpi_cache_is_enabled (
            void )
```

Return 1 if the communication cache is enabled, 0 otherwise

57.22.5.50 starpu_mpi_cache_set()

```
int starpu_mpi_cache_set (
            int enabled )
```

If `enabled` is 1, enable the communication cache. Otherwise, clean the cache if it was enabled and disable it.

57.22.5.51 starpu_mpi_cache_flush()

```
void starpu_mpi_cache_flush (
            MPI_Comm comm,
            starpu_data_handle_t data_handle )
```

Clear the send and receive communication cache for the data `data_handle` and invalidate the value. The function has to be called at the same point of task graph submission by all the MPI nodes on which the handle was registered. The function does nothing if the cache mechanism is disabled (see STARPU_MPI_CACHE).

57.22.5.52 starpu_mpi_cache_flush_all_data()

```
void starpu_mpi_cache_flush_all_data (
            MPI_Comm comm )
```

Clear the send and receive communication cache for all data and invalidate their values. The function has to be called at the same point of task graph submission by all the MPI nodes. The function does nothing if the cache mechanism is disabled (see STARPU_MPI_CACHE).

57.22.5.53 starpu_mpi_cached_receive()

```
int starpu_mpi_cached_receive (
            starpu_data_handle_t data_handle )
```

Test whether `data_handle` is cached for reception, i.e. the value was previously received from the owner node, and not flushed since then.

57.22.5.54 starpu_mpi_cached_receive_set()

```
int starpu_mpi_cached_receive_set (
            starpu_data_handle_t data )
```

If `data` is already available in the reception cache, return 1 If `data` is NOT available in the reception cache, add it to the cache and return 0 Return 0 if the communication cache is not enabled

57.22.5.55 starpu_mpi_cached_receive_clear()

```
void starpu_mpi_cached_receive_clear (
            starpu_data_handle_t data )
```

Remove `data` from the reception cache

57.22.5.56 starpu_mpi_cached_send()

```
int starpu_mpi_cached_send (
            starpu_data_handle_t data_handle,
            int dest )
```

Test whether `data_handle` is cached for emission to node `dest`, i.e. the value was previously sent to `dest`, and not flushed since then.

57.22.5.57 starpu_mpi_cached_send_set()

```
int starpu_mpi_cached_send_set (
            starpu_data_handle_t data,
            int dest )
```

If `data` is already available in the emission cache for node `dest`, return 1 If `data` is NOT available in the emission cache for node `dest`, add it to the cache and return 0 Return 0 if the communication cache is not enabled

57.22.5.58 starpu_mpi_cached_send_clear()

```
void starpu_mpi_cached_send_clear (
            starpu_data_handle_t data )
```

Remove `data` from the emission cache

57.22.5.59 starpu_mpi_data_register_comm()

```
void starpu_mpi_data_register_comm (
            starpu_data_handle_t data_handle,
            starpu_mpi_tag_t data_tag,
            int rank,
            MPI_Comm comm )
```

Register to MPI a StarPU data handle with the given tag, rank and MPI communicator. It also automatically clears the MPI communication cache when unregistering the data.

57.22.5.60 starpu_mpi_data_set_tag()

```
void starpu_mpi_data_set_tag (
            starpu_data_handle_t handle,
            starpu_mpi_tag_t data_tag )
```

Register to MPI a StarPU data handle with the given tag. No rank will be defined. It also automatically clears the MPI communication cache when unregistering the data.

57.22.5.61 starpu_mpi_data_set_rank_comm()

```
void starpu_mpi_data_set_rank_comm (
            starpu_data_handle_t handle,
            int rank,
            MPI_Comm comm )
```

Register to MPI a StarPU data handle with the given rank and given communicator. No tag will be defined. It also automatically clears the MPI communication cache when unregistering the data.

57.22.5.62 starpu_mpi_data_get_rank()

```
int starpu_mpi_data_get_rank (
            starpu_data_handle_t handle )
```

Return the rank of the given data.

57.22.5.63 starpu_mpi_data_get_tag()

```
starpu_mpi_tag_t starpu_mpi_data_get_tag (
            starpu_data_handle_t handle )
```

Return the tag of the given data.

57.22.5.64 starpu_mpi_data_get_redux_map()

```
char * starpu_mpi_data_get_redux_map (
            starpu_data_handle_t handle )
```

Return the redux map of the given data.

57.22.5.65 starpu_mpi_task_insert()

```
int starpu_mpi_task_insert (
            MPI_Comm comm,
            struct starpu_codelet * codelet,
             ... )
```

Create and submit a task corresponding to codelet with the following arguments. The argument list must be zero-terminated. The arguments following the codelet are the same types as for the function starpu_task_insert(). Access modes for data can also be set with STARPU_SSEND to specify the data has to be sent using a synchronous and non-blocking mode (see starpu_mpi_issend()). The extra argument STARPU_EXECUTE_ON_NODE followed by an integer allows to specify the MPI node to execute the codelet. It is also possible to specify that the node owning a specific data will execute the codelet, by using STARPU_EXECUTE_ON_DATA followed by a data handle.
The internal algorithm is as follows:

1. Find out which MPI node is going to execute the codelet.
    - If there is only one node owning data in STARPU_W mode, it will be selected;
    - If there is several nodes owning data in STARPU_W mode, a node will be selected according to a given node selection policy (see STARPU_NODE_SELECTION_POLICY or starpu_mpi_node_selection_set_current_policy())
    - The argument STARPU_EXECUTE_ON_NODE followed by an integer can be used to specify the node; Ignored if the node value is -1.
    - The argument STARPU_EXECUTE_ON_DATA followed by a data handle can be used to specify that the node owing the given data will execute the codelet.
2. Send and receive data as requested. Nodes owning data which need to be read by the task are sending them to the MPI node which will execute it. The latter receives them.
3. Execute the codelet. This is done by the MPI node selected in the 1st step of the algorithm.
4. If several MPI nodes own data to be written to, send written data back to their owners.

The algorithm also includes a communication cache mechanism that allows not to send data twice to the same MPI node, unless the data has been modified. The cache can be disabled (see STARPU_MPI_CACHE).

57.22.5.66 starpu_mpi_insert_task()

```
int starpu_mpi_insert_task (
            MPI_Comm comm,
            struct starpu_codelet * codelet,
             ... )
```

Identical to starpu_mpi_task_insert(). Symbol kept for backward compatibility.

57.22.5.67 starpu_mpi_task_build()

```
struct starpu_task * starpu_mpi_task_build (
            MPI_Comm comm,
            struct starpu_codelet * codelet,
             ... )
```

Create a task corresponding to `codelet` with the following given arguments. The argument list must be zero-terminated. The function performs the first two steps of the function starpu_mpi_task_insert(), i.e. submitting the MPI communications needed before the execution of the task, and the creation of the task on one node. Only the MPI node selected in the first step of the algorithm will return a valid task structure which can then be submitted, others will return `NULL`. The function starpu_mpi_task_post_build() MUST be called after that on all nodes, and after the submission of the task on the node which creates it, with the SAME list of arguments.

57.22.5.68 starpu_mpi_task_build_v()

```
struct starpu_task * starpu_mpi_task_build_v (
            MPI_Comm comm,
            struct starpu_codelet * codelet,
            va_list varg_list )
```

Offer a va_list variant of starpu_mpi_task_build.

57.22.5.69 starpu_mpi_task_post_build()

```
int starpu_mpi_task_post_build (
            MPI_Comm comm,
            struct starpu_codelet * codelet,
             ... )
```

MUST be called after a call to starpu_mpi_task_build(), with the SAME list of arguments. Perform the fourth – last – step of the algorithm described in starpu_mpi_task_insert().

57.22.5.70 starpu_mpi_task_post_build_v()

```
int starpu_mpi_task_post_build_v (
            MPI_Comm comm,
            struct starpu_codelet * codelet,
            va_list varg_list )
```

Offer a va_list variant of starpu_mpi_task_post_build.

57.22.5.71 starpu_mpi_task_exchange_data_before_execution()

```
int starpu_mpi_task_exchange_data_before_execution (
            MPI_Comm comm,
            struct starpu_task * task,
            struct starpu_data_descr * descrs,
            struct starpu_mpi_task_exchange_params * params )
```

Perform all necessary communications needed before the execution of the given task. The field `priority` of `params` will be set with the rank of the node which is selected to submit `task`. After calling this function, and the submission of the task for the selected node, all nodes MUST call the function starpu_mpi_task_exchange_data_after_execution() with the parameters `descrs` and `params`.

57.22.5.72 starpu_mpi_task_exchange_data_after_execution()

```
int starpu_mpi_task_exchange_data_after_execution (
            MPI_Comm comm,
            struct starpu_data_descr * descrs,
            unsigned nb_data,
            struct starpu_mpi_task_exchange_params params )
```

MUST be called after a call to starpu_mpi_task_exchange_data_before_execution() with the same arguments `descrs` and `params`. `nb_data` is the number of data in `descrs`. Perform all the necessary communications needed after the execution of the task, i.e the fourth – last – step of the algorithm described in starpu_mpi_task_insert().

57.22.5.73 starpu_mpi_get_data_on_node()

```
int starpu_mpi_get_data_on_node (
            MPI_Comm comm,
            starpu_data_handle_t data_handle,
            int node )
```

Transfer data `data_handle` to MPI node `node`, sending it from its owner if needed. At least the target node and the owner have to call the function. This waits for the transfer to be over.

57.22.5.74 starpu_mpi_get_data_on_node_detached()

```
int starpu_mpi_get_data_on_node_detached (
            MPI_Comm comm,
            starpu_data_handle_t data_handle,
            int node,
            void(*)(void *) callback,
            void * arg )
```

Transfer data `data_handle` to MPI node `node`, sending it from its owner if needed. At least the target node and the owner have to call the function. On reception, the `callback` function is called with the argument `arg`.

57.22.5.75 starpu_mpi_get_data_on_all_nodes_detached()

```
void starpu_mpi_get_data_on_all_nodes_detached (
            MPI_Comm comm,
            starpu_data_handle_t data_handle )
```

Transfer data `data_handle` to all MPI nodes, sending it from its owner if needed. All nodes have to call the function.

57.22.5.76 starpu_mpi_data_migrate()

```
void starpu_mpi_data_migrate (
            MPI_Comm comm,
            starpu_data_handle_t handle,
            int new_rank )
```

Submit migration of the data onto the `new_rank` MPI node. This means both submitting the transfer of the data to node `new_rank` if it hasn't been submitted already, and setting the home node of the data to the new node. Further data transfers submitted by starpu_mpi_task_insert() will be done from that new node. This function thus needs to be called on all nodes which have registered the data at the same point of tasks submissions. This also flushes the cache for this data to avoid incoherencies.

57.22.5.77 starpu_mpi_node_selection_register_policy()

```
int starpu_mpi_node_selection_register_policy (
            starpu_mpi_select_node_policy_func_t policy_func )
```

Register a new policy which can then be used when there is several nodes owning data in STARPU_W mode. Here an example of function defining a node selection policy. The codelet will be executed on the node owing the first data with a size bigger than 1M, or on the node 0 if no data fits the given size.

```
int my_node_selection_policy(int me, int nb_nodes, struct starpu_data_descr *descr, int nb_data)
{
     // me is the current MPI rank
     // nb_nodes is the number of MPI nodes
     // descr is the description of the data specified when calling starpu_mpi_task_insert
     // nb_data is the number of data in descr
     int i;
     for(i= 0 ; i<nb_data ; i++)
     {
        starpu_data_handle_t data = descr[i].handle;
        enum starpu_data_access_mode mode = descr[i].mode;
        if (mode & STARPU_R)
        {
                int rank = starpu_data_get_rank(data);
                size_t size = starpu_data_get_size(data);
                if (size > 1024*1024) return rank;
        }
     }
     return 0;
     }
```

57.22.5.78 starpu_mpi_node_selection_unregister_policy()

```
int starpu_mpi_node_selection_unregister_policy (
            int policy )
```

Unregister a previously registered policy.

57.22.5.79 starpu_mpi_node_selection_get_current_policy()

```
int starpu_mpi_node_selection_get_current_policy (
            void  )
```

Return the current policy used to select the node which will execute the codelet

57.22.5.80 starpu_mpi_node_selection_set_current_policy()

```
int starpu_mpi_node_selection_set_current_policy (
            int policy )
```

Set the current policy used to select the node which will execute the codelet. The policy STARPU_MPI_NODE_SELECTION_MOST_R selects the node having the most data in STARPU_R mode so as to minimize the amount of data to be transferred.

57.22.5.81 starpu_mpi_redux_data()

```
int starpu_mpi_redux_data (
            MPI_Comm comm,
            starpu_data_handle_t data_handle )
```

Perform a reduction on the given data `handle`. All nodes send the data to its owner node which will perform a reduction.

57.22.5.82 starpu_mpi_redux_data_prio()

```
int starpu_mpi_redux_data_prio (
            MPI_Comm comm,
            starpu_data_handle_t data_handle,
            int prio )
```

Similar to starpu_mpi_redux_data(), but take a priority `prio`.

57.22.5.83 starpu_mpi_redux_data_tree()

```
int starpu_mpi_redux_data_tree (
            MPI_Comm comm,
            starpu_data_handle_t data_handle,
            int arity )
```

Perform a reduction on the given data `handle`. Nodes perform the reduction through in a tree-based fashion. The tree use is an `arity` - ary tree.

57.22.5.84 starpu_mpi_redux_data_prio_tree()

```
int starpu_mpi_redux_data_prio_tree (
            MPI_Comm comm,
            starpu_data_handle_t data_handle,
            int prio,
            int arity )
```

Similar to starpu_mpi_redux_data_tree(), but take a priority `prio`.

57.22.5.85 starpu_mpi_scatter_detached()

```
int starpu_mpi_scatter_detached (
            starpu_data_handle_t * data_handles,
            int count,
            int root,
            MPI_Comm comm,
            void(*)(void *) scallback,
            void * sarg,
            void(*)(void *) rcallback,
            void * rarg )
```

Scatter data among processes of the communicator based on the ownership of the data. For each data of the array `data_handles`, the process `root` sends the data to the process owning this data. Processes receiving data must have valid data handles to receive them. On completion of the collective communication, the `scallback` function is called with the argument `sarg` on the process `root`, the `rcallback` function is called with the argument `rarg` on any other process.

57.22.5.86 starpu_mpi_gather_detached()

```
int starpu_mpi_gather_detached (
            starpu_data_handle_t * data_handles,
            int count,
            int root,
            MPI_Comm comm,
            void(*)(void *) scallback,
            void * sarg,
            void(*)(void *) rcallback,
            void * rarg )
```

Gather data from the different processes of the communicator onto the process `root`. Each process owning data handle in the array `data_handles` will send them to the process `root`. The process `root` must have valid data handles to receive the data. On completion of the collective communication, the `rcallback` function is called with the argument `rarg` on the process root, the `scallback` function is called with the argument `sarg` on any other process.

57.22.5.87 starpu_mpi_coop_sends_set_use()

```
void starpu_mpi_coop_sends_set_use (
            int use_coop_sends )
```

Enable or disable coop sends.
Used for benchmark, not recommended for production: can cause problems if there are still communications while disabling, or when shutting down StarPU.
This function must be called after the initialization of StarPU.

57.22.5.88 starpu_mpi_coop_sends_get_use()

```
int starpu_mpi_coop_sends_get_use (
            void )
```

Return whether coop sends are enabled or not.

57.22.5.89 starpu_mpi_coop_sends_data_handle_nb_sends()

```
void starpu_mpi_coop_sends_data_handle_nb_sends (
            starpu_data_handle_t data_handle,
            int nb_sends )
```

Explicit the number of different sends of the `data_handle`. When the number of sends is reached, a collective operation is triggered. If this function isn't called, StarPU will trigger a collective operation containing only posted sends while the data wasn't available.

57.22.5.90 starpu_mpi_comm_stats_disable()

```
void starpu_mpi_comm_stats_disable (
            void )
```

Disable the aggregation of communications statistics.

57.22.5.91 starpu_mpi_comm_stats_enable()

```
void starpu_mpi_comm_stats_enable (
            void )
```

Enable the aggregation of communications statistics.

57.22.5.92 starpu_mpi_comm_stats_retrieve()

```
void starpu_mpi_comm_stats_retrieve (
            size_t * comm_stats )
```

Retrieve the current communications statistics from the current node in the array `comm_stats` which must have a size greater or equal to the world size. Communications statistics must have been enabled, either through the function starpu_mpi_comm_stats_enable() or through the environment variable STARPU_MPI_STATS.

57.22.5.93 starpu_mpi_data_cpy()

```
int starpu_mpi_data_cpy (
            starpu_data_handle_t dst_handle,
            starpu_data_handle_t src_handle,
            MPI_Comm comm,
            int asynchronous,
            void(*)(void *) callback_func,
            void * callback_arg )
```

Copy the content of `src_handle` into `dst_handle`. If both data are on the same node, the function starpu_data_cpy() is called, otherwise a MPI transfer is initiated between both nodes. The parameter `asynchronous` indicates whether the function should block or not. If `callback_func` is not `NULL`, this callback function is executed on the owner node of the data `dst_handle` after the handle has been received, and it is given the pointer `callback_arg` as argument. See Other MPI Utility Functions for more details.

57.22.5.94 starpu_mpi_data_cpy_priority()

```
int starpu_mpi_data_cpy_priority (
            starpu_data_handle_t dst_handle,
            starpu_data_handle_t src_handle,
            MPI_Comm comm,
            int asynchronous,
            void(*)(void *) callback_func,
            void * callback_arg,
            int priority )
```

Similar to starpu_mpi_data_cpy(), but take a priority `prio`.

57.22.5.95 starpu_mpi_tags_allocate()

```
int64_t starpu_mpi_tags_allocate (
            int64_t nbtags )
```

Book a range of unique tags of size `nbtags` to be used to register StarPU data handles. This function returns the minimal tag value available `mintag` to allow the registration of data with tags in the continuous range [[`mintag`, `mintag` + `nbtags`]]

Note that this function must be called by all MPI processes involved in the computations with the same parameters and in the exact same order to make sure the tags are identical from one node to another.

57.22.5.96 starpu_mpi_tags_free()

```
void starpu_mpi_tags_free (
            int64_t mintag )
```

Release the range of tags starting by the given `mintag` value. The mintag value must be a value obtained through a call to starpu_mpi_tags_allocate().

Note that this function must be called by all MPI processes involved in the computations with the same parameters and in the exact same order to make sure the tags are identical from one node to another as for starpu_mpi_tags_allocate().

57.22.6 Variable Documentation

57.22.6.1 do_execute

`int starpu_mpi_task_exchange_params::do_execute`
is the caller going to execute the task

57.22.6.2 xrank

`int starpu_mpi_task_exchange_params::xrank`
node executing the task

57.22.6.3 priority

`int starpu_mpi_task_exchange_params::priority`
priority of the task being executed

57.23 OpenCL Extensions

Data Structures

- struct starpu_opencl_program

Macros

- #define STARPU_USE_OPENCL
- #define STARPU_OPENCL_DATADIR
- #define STARPU_MAXOPENCLDEVS

Writing OpenCL kernels

- void starpu_opencl_get_context (int devid, cl_context ∗context)
- void starpu_opencl_get_device (int devid, cl_device_id ∗device)
- void starpu_opencl_get_queue (int devid, cl_command_queue ∗queue)
- void starpu_opencl_get_current_context (cl_context ∗context)
- void starpu_opencl_get_current_queue (cl_command_queue ∗queue)
- int starpu_opencl_set_kernel_args (cl_int ∗err, cl_kernel ∗kernel,...)

Compiling OpenCL kernels

Source codes for OpenCL kernels can be stored in a file or in a string. StarPU provides functions to build the program executable for each available OpenCL device as a cl_program object. This program executable can then be loaded within a specific queue as explained in the next section. These are only helpers, Applications can also fill a starpu_opencl_program array by hand for more advanced use (e.g. different programs on the different OpenCL devices, for relocation purpose for instance).

- void starpu_opencl_load_program_source (const char ∗source_file_name, char ∗located_file_name, char ∗located_dir_name, char ∗opencl_program_source)
- void starpu_opencl_load_program_source_malloc (const char ∗source_file_name, char ∗∗located_file_name, char ∗∗located_dir_name, char ∗∗opencl_program_source)
- int starpu_opencl_compile_opencl_from_file (const char ∗source_file_name, const char ∗build_options)
- int starpu_opencl_compile_opencl_from_string (const char ∗opencl_program_source, const char ∗file_name, const char ∗build_options)
- int starpu_opencl_load_binary_opencl (const char ∗kernel_id, struct starpu_opencl_program ∗opencl_programs)
- int starpu_opencl_load_opencl_from_file (const char ∗source_file_name, struct starpu_opencl_program ∗opencl_programs, const char ∗build_options)
- int starpu_opencl_load_opencl_from_string (const char ∗opencl_program_source, struct starpu_opencl_program ∗opencl_programs, const char ∗build_options)
- int starpu_opencl_unload_opencl (struct starpu_opencl_program ∗opencl_programs)

Loading OpenCL kernels

- int starpu_opencl_load_kernel (cl_kernel ∗kernel, cl_command_queue ∗queue, struct starpu_opencl_program ∗opencl_programs, const char ∗kernel_name, int devid)
- int starpu_opencl_release_kernel (cl_kernel kernel)

OpenCL Statistics

- int starpu_opencl_collect_stats (cl_event event)

OpenCL Utilities

- const char ∗ starpu_opencl_error_string (cl_int status)
- void starpu_opencl_display_error (const char ∗func, const char ∗file, int line, const char ∗msg, cl_int status)
- static __starpu_inline void starpu_opencl_report_error (const char ∗func, const char ∗file, int line, const char ∗msg, cl_int status)
- cl_int starpu_opencl_allocate_memory (int devid, cl_mem ∗addr, size_t size, cl_mem_flags flags)
- cl_int starpu_opencl_copy_ram_to_opencl (void ∗ptr, unsigned src_node, cl_mem buffer, unsigned dst_node, size_t size, size_t offset, cl_event ∗event, int ∗ret)
- cl_int starpu_opencl_copy_opencl_to_ram (cl_mem buffer, unsigned src_node, void ∗ptr, unsigned dst_node, size_t size, size_t offset, cl_event ∗event, int ∗ret)
- cl_int starpu_opencl_copy_opencl_to_opencl (cl_mem src, unsigned src_node, size_t src_offset, cl_mem dst, unsigned dst_node, size_t dst_offset, size_t size, cl_event ∗event, int ∗ret)
- cl_int starpu_opencl_copy_async_sync (uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, size_t dst_offset, unsigned dst_node, size_t size, cl_event ∗event)
- #define STARPU_OPENCL_DISPLAY_ERROR(status)
- #define STARPU_OPENCL_REPORT_ERROR(status)
- #define STARPU_OPENCL_REPORT_ERROR_WITH_MSG(msg, status)

57.23.1 Detailed Description

57.23.2 Data Structure Documentation

57.23.2.1 struct starpu_opencl_program

Store the OpenCL programs as compiled for the different OpenCL devices.

Data Fields

cl_program	programs[STARPU_MAXOPENCLDEVS]	Store each program for each OpenCL device.

57.23.3 Macro Definition Documentation

57.23.3.1 STARPU_USE_OPENCL

`#define STARPU_USE_OPENCL`

Defined when StarPU has been installed with OpenCL support. It should be used in your code to detect the availability of OpenCL as shown in Full source code for the 'Scaling a Vector' example.

57.23.3.2 STARPU_OPENCL_DATADIR

`#define STARPU_OPENCL_DATADIR`

Define the directory in which the OpenCL codelets of the applications provided with StarPU have been installed.

57.23.3.3 STARPU_MAXOPENCLDEVS

`#define STARPU_MAXOPENCLDEVS`

Define the maximum number of OpenCL devices that are supported by StarPU.

57.23.3.4 STARPU_OPENCL_DISPLAY_ERROR

```
#define STARPU_OPENCL_DISPLAY_ERROR(
                status )
```

Call the function starpu_opencl_display_error() with the error `status`, the current function name, current file and line number, and a empty message.

57.23.3.5 STARPU_OPENCL_REPORT_ERROR

```
#define STARPU_OPENCL_REPORT_ERROR(
              status )
```

Call the function starpu_opencl_report_error() with the error `status`, the current function name, current file and line number, and a empty message.

57.23.3.6 STARPU_OPENCL_REPORT_ERROR_WITH_MSG

```
#define STARPU_OPENCL_REPORT_ERROR_WITH_MSG(
              msg,
              status )
```

Call the function starpu_opencl_report_error() with `msg` and `status`, the current function name, current file and line number.

57.23.4 Function Documentation

57.23.4.1 starpu_opencl_get_context()

```
void starpu_opencl_get_context (
             int devid,
             cl_context * context )
```

Return the OpenCL context of the device designated by `devid` in `context`. See OpenCL Support for more details.

57.23.4.2 starpu_opencl_get_device()

```
void starpu_opencl_get_device (
             int devid,
             cl_device_id * device )
```

Return the cl_device_id corresponding to `devid` in `device`. See OpenCL Support for more details.

57.23.4.3 starpu_opencl_get_queue()

```
void starpu_opencl_get_queue (
             int devid,
             cl_command_queue * queue )
```

Return the command queue of the device designated by `devid` into `queue`. See OpenCL Support for more details.

57.23.4.4 starpu_opencl_get_current_context()

```
void starpu_opencl_get_current_context (
             cl_context * context )
```

Return the context of the current worker. See OpenCL Support for more details.

57.23.4.5 starpu_opencl_get_current_queue()

```
void starpu_opencl_get_current_queue (
             cl_command_queue * queue )
```

Return the computation kernel command queue of the current worker. See OpenCL Support for more details.

57.23.4.6 starpu_opencl_set_kernel_args()

```
int starpu_opencl_set_kernel_args (
             cl_int * err,
             cl_kernel * kernel,
              ... )
```

Set the arguments of a given kernel. The list of arguments must be given as `(size_t size_of_the_argument, cl_mem * pointer_to_the_argument)`. The last argument must be 0. Return the number of arguments that were successfully set. In case of failure, return the id of the argument that could not be set and `err` is set to the error returned by OpenCL. Otherwise, return the number of arguments that were set.
Here an example:

```
int n;
cl_int err;
cl_kernel kernel;
n = starpu_opencl_set_kernel_args(&err, 2, &kernel, sizeof(foo), &foo, sizeof(bar), &bar, 0);
if (n != 2) fprintf(stderr, "Error : %d\n", err);
```

See OpenCL Support for more details.

57.23.4.7 starpu_opencl_load_program_source()

```
void starpu_opencl_load_program_source (
             const char * source_file_name,
             char * located_file_name,
             char * located_dir_name,
             char * opencl_program_source )
```

Store the contents of the file `source_file_name` in the buffer `opencl_program_source`. The file `source_file_name` can be located in the current directory, or in the directory specified by the environment variable STARPU_OPENCL_PROGRAM_DIR, or in the directory `share/starpu/opencl` of the installation directory of StarPU, or in the source directory of StarPU. When the file is found, `located_file_name` is the full name of the file as it has been located on the system, `located_dir_name` the directory where it has been located. Otherwise, they are both set to the empty string. See OpenCL Support for more details.

57.23.4.8 starpu_opencl_load_program_source_malloc()

```
void starpu_opencl_load_program_source_malloc (
             const char * source_file_name,
             char ** located_file_name,
             char ** located_dir_name,
             char ** opencl_program_source )
```

Similar to function starpu_opencl_load_program_source() but allocate the buffers `located_file_name`, `located_dir_name` and `opencl_program_source`. See OpenCL Support for more details.

57.23.4.9 starpu_opencl_compile_opencl_from_file()

```
int starpu_opencl_compile_opencl_from_file (
             const char * source_file_name,
             const char * build_options )
```

Compile the OpenCL kernel stored in the file `source_file_name` with the given options `build_options` and store the result in the directory `$STARPU_HOME/.starpu/opencl` with the same filename as `source_file_name`. The compilation is done for every OpenCL device, and the filename is suffixed with the vendor id and the device id of the OpenCL device. See OpenCL Support for more details.

57.23.4.10 starpu_opencl_compile_opencl_from_string()

```
int starpu_opencl_compile_opencl_from_string (
             const char * opencl_program_source,
             const char * file_name,
             const char * build_options )
```

Compile the OpenCL kernel in the string `opencl_program_source` with the given options `build_options` and store the result in the directory `$STARPU_HOME/.starpu/opencl` with the filename `file_name`. The compilation is done for every OpenCL device, and the filename is suffixed with the vendor id and the device id of the OpenCL device. See OpenCL Support for more details.

57.23.4.11 starpu_opencl_load_binary_opencl()

```
int starpu_opencl_load_binary_opencl (
             const char * kernel_id,
            struct starpu_opencl_program * opencl_programs )
```

Compile the binary OpenCL kernel identified with `kernel_id`. For every OpenCL device, the binary OpenCL kernel will be loaded from the file `$STARPU_HOME/.starpu/opencl/<kernel_id>.<device_type>.vendor_id_<vendor_id>`*device_id*`<device_id>`. See OpenCL Support for more details.

57.23.4.12 starpu_opencl_load_opencl_from_file()

```
int starpu_opencl_load_opencl_from_file (
            const char * source_file_name,
            struct starpu_opencl_program * opencl_programs,
            const char * build_options )
```

Compile an OpenCL source code stored in a file. See OpenCL Support for more details.

57.23.4.13 starpu_opencl_load_opencl_from_string()

```
int starpu_opencl_load_opencl_from_string (
            const char * opencl_program_source,
            struct starpu_opencl_program * opencl_programs,
            const char * build_options )
```

Compile an OpenCL source code stored in a string. See OpenCL Support for more details.

57.23.4.14 starpu_opencl_unload_opencl()

```
int starpu_opencl_unload_opencl (
            struct starpu_opencl_program * opencl_programs )
```

Unload an OpenCL compiled code. See OpenCL Support for more details.

57.23.4.15 starpu_opencl_load_kernel()

```
int starpu_opencl_load_kernel (
            cl_kernel * kernel,
            cl_command_queue * queue,
            struct starpu_opencl_program * opencl_programs,
            const char * kernel_name,
            int devid )
```

Create a kernel `kernel` for device `devid`, on its computation command queue returned in `queue`, using program `opencl_programs` and name `kernel_name`. See OpenCL Support for more details.

57.23.4.16 starpu_opencl_release_kernel()

```
int starpu_opencl_release_kernel (
            cl_kernel kernel )
```

Release the given `kernel`, to be called after kernel execution. See OpenCL Support for more details.

57.23.4.17 starpu_opencl_collect_stats()

```
int starpu_opencl_collect_stats (
            cl_event event )
```

Collect statistics on a kernel execution. After termination of the kernels, the OpenCL codelet should call this function with the event returned by `clEnqueueNDRangeKernel()`, to let StarPU collect statistics about the kernel execution (used cycles, consumed energy). See OpenCL-specific Optimizations for more details.

57.23.4.18 starpu_opencl_error_string()

```
const char * starpu_opencl_error_string (
            cl_int status )
```

Return the error message in English corresponding to `status`, an OpenCL error code. See OpenCL Support for more details.

57.23.4.19 starpu_opencl_display_error()

```
void starpu_opencl_display_error (
            const char * func,
            const char * file,
            int line,
            const char * msg,
            cl_int status )
```

Given a valid error status, print the corresponding error message on `stdout`, along with the function name `func`, the filename `file`, the line number `line` and the message `msg`. See OpenCL Support for more details.

57.23.4.20 starpu_opencl_report_error()

```
static __starpu_inline void starpu_opencl_report_error (
            const char * func,
            const char * file,
            int line,
            const char * msg,
            cl_int status ) [static]
```

Call the function starpu_opencl_display_error() and abort.

57.23.4.21 starpu_opencl_allocate_memory()

```
cl_int starpu_opencl_allocate_memory (
            int devid,
            cl_mem * addr,
            size_t size,
            cl_mem_flags flags )
```

Allocate `size` bytes of memory, stored in `addr`. `flags` must be a valid combination of `cl_mem_flags` values. See Data allocation for more details.

57.23.4.22 starpu_opencl_copy_ram_to_opencl()

```
cl_int starpu_opencl_copy_ram_to_opencl (
            void * ptr,
            unsigned src_node,
            cl_mem buffer,
            unsigned dst_node,
            size_t size,
            size_t offset,
            cl_event * event,
            int * ret )
```

Copy `size` bytes from the given `ptr` on RAM `src_node` to the given `buffer` on OpenCL `dst_node`. `offset` is the offset, in bytes, in `buffer`. if `event` is `NULL`, the copy is synchronous, i.e the queue is synchronised before returning. If not `NULL`, `event` can be used after the call to wait for this particular copy to complete. This function returns `CL_SUCCESS` if the copy was successful, or a valid OpenCL error code otherwise. The integer pointed to by `ret` is set to `-EAGAIN` if the asynchronous launch was successful, or to 0 if `event` was `NULL`. See Data copy for more details.

57.23.4.23 starpu_opencl_copy_opencl_to_ram()

```
cl_int starpu_opencl_copy_opencl_to_ram (
            cl_mem buffer,
            unsigned src_node,
            void * ptr,
            unsigned dst_node,
            size_t size,
            size_t offset,
            cl_event * event,
            int * ret )
```

Copy `size` bytes asynchronously from the given `buffer` on OpenCL `src_node` to the given `ptr` on RAM `dst_node`. `offset` is the offset, in bytes, in `buffer`. if `event` is `NULL`, the copy is synchronous, i.e the queue is synchronised before returning. If not `NULL`, `event` can be used after the call to wait for this particular copy to complete. This function returns `CL_SUCCESS` if the copy was successful, or a valid OpenCL error code otherwise. The integer pointed to by `ret` is set to `-EAGAIN` if the asynchronous launch was successful, or to 0 if `event` was `NULL`. See Data copy for more details.

57.23.4.24 starpu_opencl_copy_opencl_to_opencl()

```
cl_int starpu_opencl_copy_opencl_to_opencl (
            cl_mem src,
            unsigned src_node,
            size_t src_offset,
            cl_mem dst,
            unsigned dst_node,
            size_t dst_offset,
            size_t size,
            cl_event * event,
            int * ret )
```

Copy `size` bytes asynchronously from byte offset `src_offset` of `src` on OpenCL `src_node` to byte offset `dst_offset` of `dst` on OpenCL `dst_node`. if `event` is `NULL`, the copy is synchronous, i.e. the queue is synchronised before returning. If not `NULL`, `event` can be used after the call to wait for this particular copy to complete. This function returns `CL_SUCCESS` if the copy was successful, or a valid OpenCL error code otherwise. The integer pointed to by `ret` is set to `-EAGAIN` if the asynchronous launch was successful, or to 0 if `event` was `NULL`. See Data copy for more details.

57.23.4.25 starpu_opencl_copy_async_sync()

```
cl_int starpu_opencl_copy_async_sync (
            uintptr_t src,
            size_t src_offset,
            unsigned src_node,
            uintptr_t dst,
            size_t dst_offset,
            unsigned dst_node,
            size_t size,
            cl_event * event )
```

Copy `size` bytes from byte offset `src_offset` of `src` on `src_node` to byte offset `dst_offset` of `dst` on `dst_node`. if `event` is `NULL`, the copy is synchronous, i.e. the queue is synchronised before returning. If not `NULL`, `event` can be used after the call to wait for this particular copy to complete. The function returns `-EAGAIN` if the asynchronous launch was successful. It returns 0 if the synchronous copy was successful, or fails otherwise. See Data copy for more details.

57.24 OpenMP Runtime Support

API for implementing OpenMP runtimes on top of StarPU.

Data Structures

- struct starpu_omp_lock_t
- struct starpu_omp_nest_lock_t
- struct starpu_omp_parallel_region_attr
- struct starpu_omp_task_region_attr

Macros

- #define STARPU_OPENMP
- #define **__STARPU_OMP_NOTHROW**

Enumerations

- enum starpu_omp_sched_value {
  starpu_omp_sched_undefined , starpu_omp_sched_static , starpu_omp_sched_dynamic , starpu_omp_sched_guided ,
  starpu_omp_sched_auto , starpu_omp_sched_runtime }
- enum starpu_omp_proc_bind_value {
  starpu_omp_proc_bind_undefined , starpu_omp_proc_bind_false , starpu_omp_proc_bind_true , starpu_omp_proc_bind_master ,
  starpu_omp_proc_bind_close , starpu_omp_proc_bind_spread }

Initialisation

- int starpu_omp_init (void) __STARPU_OMP_NOTHROW
- void starpu_omp_shutdown (void) __STARPU_OMP_NOTHROW

Parallel

- void starpu_omp_parallel_region (const struct starpu_omp_parallel_region_attr ∗attr) __STARPU_OMP_NOTHROW
- void starpu_omp_master (void(∗f)(void ∗arg), void ∗arg) __STARPU_OMP_NOTHROW
- int starpu_omp_master_inline (void) __STARPU_OMP_NOTHROW

Synchronization

- void starpu_omp_barrier (void) __STARPU_OMP_NOTHROW
- void starpu_omp_critical (void(∗f)(void ∗arg), void ∗arg, const char ∗name) __STARPU_OMP_NOTHROW
- void starpu_omp_critical_inline_begin (const char ∗name) __STARPU_OMP_NOTHROW
- void starpu_omp_critical_inline_end (const char ∗name) __STARPU_OMP_NOTHROW

Worksharing

- void starpu_omp_single (void(∗f)(void ∗arg), void ∗arg, int nowait) __STARPU_OMP_NOTHROW
- int starpu_omp_single_inline (void) __STARPU_OMP_NOTHROW
- void starpu_omp_single_copyprivate (void(∗f)(void ∗arg, void ∗data, unsigned long long data_size), void ∗arg, void ∗data, unsigned long long data_size) __STARPU_OMP_NOTHROW
- void ∗ starpu_omp_single_copyprivate_inline_begin (void ∗data) __STARPU_OMP_NOTHROW
- void starpu_omp_single_copyprivate_inline_end (void) __STARPU_OMP_NOTHROW
- void starpu_omp_for (void(∗f)(unsigned long long _first_i, unsigned long long _nb_i, void ∗arg), void ∗arg, unsigned long long nb_iterations, unsigned long long chunk, int schedule, int ordered, int nowait) __STARPU_OMP_NOTHROW

- int starpu_omp_for_inline_first (unsigned long long nb_iterations, unsigned long long chunk, int schedule, int ordered, unsigned long long ∗_first_i, unsigned long long ∗_nb_i) __STARPU_OMP_NOTHROW
- int starpu_omp_for_inline_next (unsigned long long nb_iterations, unsigned long long chunk, int schedule, int ordered, unsigned long long ∗_first_i, unsigned long long ∗_nb_i) __STARPU_OMP_NOTHROW
- void starpu_omp_for_alt (void(∗f)(unsigned long long _begin_i, unsigned long long _end_i, void ∗arg), void ∗arg, unsigned long long nb_iterations, unsigned long long chunk, int schedule, int ordered, int nowait) __↩ STARPU_OMP_NOTHROW
- int starpu_omp_for_inline_first_alt (unsigned long long nb_iterations, unsigned long long chunk, int schedule, int ordered, unsigned long long ∗_begin_i, unsigned long long ∗_end_i) __STARPU_OMP_NOTHROW
- int starpu_omp_for_inline_next_alt (unsigned long long nb_iterations, unsigned long long chunk, int schedule, int ordered, unsigned long long ∗_begin_i, unsigned long long ∗_end_i) __STARPU_OMP_NOTHROW
- void starpu_omp_ordered (void(∗f)(void ∗arg), void ∗arg) __STARPU_OMP_NOTHROW
- void starpu_omp_ordered_inline_begin (void) __STARPU_OMP_NOTHROW
- void starpu_omp_ordered_inline_end (void) __STARPU_OMP_NOTHROW
- void starpu_omp_sections (unsigned long long nb_sections, void(∗∗section_f)(void ∗arg), void ∗∗section_arg, int nowait) __STARPU_OMP_NOTHROW
- void starpu_omp_sections_combined (unsigned long long nb_sections, void(∗section_f)(unsigned long long section_num, void ∗arg), void ∗section_arg, int nowait) __STARPU_OMP_NOTHROW

Task

- void starpu_omp_task_region (const struct starpu_omp_task_region_attr ∗attr) __STARPU_OMP_↩ NOTHROW
- void starpu_omp_taskwait (void) __STARPU_OMP_NOTHROW
- void starpu_omp_taskgroup (void(∗f)(void ∗arg), void ∗arg) __STARPU_OMP_NOTHROW
- void starpu_omp_taskgroup_inline_begin (void) __STARPU_OMP_NOTHROW
- void starpu_omp_taskgroup_inline_end (void) __STARPU_OMP_NOTHROW
- void **starpu_omp_taskloop_inline_begin** (struct starpu_omp_task_region_attr ∗attr) __STARPU_OMP_↩ NOTHROW
- void **starpu_omp_taskloop_inline_end** (const struct starpu_omp_task_region_attr ∗attr) __STARPU_↩ OMP_NOTHROW

API

- void starpu_omp_set_num_threads (int threads) __STARPU_OMP_NOTHROW
- int starpu_omp_get_num_threads (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_thread_num (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_max_threads (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_num_procs (void) __STARPU_OMP_NOTHROW
- int starpu_omp_in_parallel (void) __STARPU_OMP_NOTHROW
- void starpu_omp_set_dynamic (int dynamic_threads) __STARPU_OMP_NOTHROW
- int starpu_omp_get_dynamic (void) __STARPU_OMP_NOTHROW
- void starpu_omp_set_nested (int nested) __STARPU_OMP_NOTHROW
- int starpu_omp_get_nested (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_cancellation (void) __STARPU_OMP_NOTHROW
- void starpu_omp_set_schedule (enum starpu_omp_sched_value kind, int modifier) __STARPU_OMP_↩ NOTHROW
- void starpu_omp_get_schedule (enum starpu_omp_sched_value ∗kind, int ∗modifier) __STARPU_OMP_↩ NOTHROW
- int starpu_omp_get_thread_limit (void) __STARPU_OMP_NOTHROW
- void starpu_omp_set_max_active_levels (int max_levels) __STARPU_OMP_NOTHROW
- int starpu_omp_get_max_active_levels (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_level (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_ancestor_thread_num (int level) __STARPU_OMP_NOTHROW
- int starpu_omp_get_team_size (int level) __STARPU_OMP_NOTHROW
- int starpu_omp_get_active_level (void) __STARPU_OMP_NOTHROW

- int starpu_omp_in_final (void) __STARPU_OMP_NOTHROW
- enum starpu_omp_proc_bind_value starpu_omp_get_proc_bind (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_num_places (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_place_num_procs (int place_num) __STARPU_OMP_NOTHROW
- void starpu_omp_get_place_proc_ids (int place_num, int ∗ids) __STARPU_OMP_NOTHROW
- int starpu_omp_get_place_num (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_partition_num_places (void) __STARPU_OMP_NOTHROW
- void starpu_omp_get_partition_place_nums (int ∗place_nums) __STARPU_OMP_NOTHROW
- void starpu_omp_set_default_device (int device_num) __STARPU_OMP_NOTHROW
- int starpu_omp_get_default_device (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_num_devices (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_num_teams (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_team_num (void) __STARPU_OMP_NOTHROW
- int starpu_omp_is_initial_device (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_initial_device (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_max_task_priority (void) __STARPU_OMP_NOTHROW
- void starpu_omp_init_lock (starpu_omp_lock_t ∗lock) __STARPU_OMP_NOTHROW
- void starpu_omp_destroy_lock (starpu_omp_lock_t ∗lock) __STARPU_OMP_NOTHROW
- void starpu_omp_set_lock (starpu_omp_lock_t ∗lock) __STARPU_OMP_NOTHROW
- void starpu_omp_unset_lock (starpu_omp_lock_t ∗lock) __STARPU_OMP_NOTHROW
- int starpu_omp_test_lock (starpu_omp_lock_t ∗lock) __STARPU_OMP_NOTHROW
- void starpu_omp_init_nest_lock (starpu_omp_nest_lock_t ∗lock) __STARPU_OMP_NOTHROW
- void starpu_omp_destroy_nest_lock (starpu_omp_nest_lock_t ∗lock) __STARPU_OMP_NOTHROW
- void starpu_omp_set_nest_lock (starpu_omp_nest_lock_t ∗lock) __STARPU_OMP_NOTHROW
- void starpu_omp_unset_nest_lock (starpu_omp_nest_lock_t ∗lock) __STARPU_OMP_NOTHROW
- int starpu_omp_test_nest_lock (starpu_omp_nest_lock_t ∗lock) __STARPU_OMP_NOTHROW
- void starpu_omp_atomic_fallback_inline_begin (void) __STARPU_OMP_NOTHROW
- void starpu_omp_atomic_fallback_inline_end (void) __STARPU_OMP_NOTHROW
- double starpu_omp_get_wtime (void) __STARPU_OMP_NOTHROW
- double starpu_omp_get_wtick (void) __STARPU_OMP_NOTHROW
- void starpu_omp_vector_annotate (starpu_data_handle_t handle, uint32_t slice_base) __STARPU_OMP_↩ NOTHROW
- struct starpu_arbiter ∗ starpu_omp_get_default_arbiter (void) __STARPU_OMP_NOTHROW
- void starpu_omp_handle_register (starpu_data_handle_t handle) __STARPU_OMP_NOTHROW
- void starpu_omp_handle_unregister (starpu_data_handle_t handle) __STARPU_OMP_NOTHROW
- starpu_data_handle_t starpu_omp_data_lookup (const void ∗ptr) __STARPU_OMP_NOTHROW

57.24.1 Detailed Description

API for implementing OpenMP runtimes on top of StarPU.

57.24.2 Data Structure Documentation

57.24.2.1 struct starpu_omp_lock_t

Opaque Simple Lock object () for inter-task synchronization operations.

See also

starpu_omp_init_lock()

starpu_omp_destroy_lock()

starpu_omp_set_lock()

starpu_omp_unset_lock()

starpu_omp_test_lock()

Data Fields

void ∗	internal	opaque pointer for internal use

57.24.2.2 struct starpu_omp_nest_lock_t

Opaque Nestable Lock object () for inter-task synchronization operations.

See also

starpu_omp_init_nest_lock()
starpu_omp_destroy_nest_lock()
starpu_omp_set_nest_lock()
starpu_omp_unset_nest_lock()
starpu_omp_test_nest_lock()

Data Fields

void ∗	internal	opaque pointer for internal use

57.24.2.3 struct starpu_omp_parallel_region_attr

Set of attributes used for creating a new parallel region.

See also

starpu_omp_parallel_region()

Data Fields

struct starpu_codelet	cl	starpu_codelet (Codelet And Tasks) to use for the parallel region implicit tasks. The codelet must provide a CPU implementation function.
starpu_data_handle_t ∗	handles	Array of zero or more starpu_data_handle_t data handle to be passed to the parallel region implicit tasks.
void ∗	cl_arg	Optional pointer to an inline argument to be passed to the region implicit tasks.
size_t	cl_arg_size	Size of the optional inline argument to be passed to the region implicit tasks, or 0 if unused.
unsigned	cl_arg_free	Boolean indicating whether the optional inline argument should be automatically freed (true), or not (false).
int	if_clause	Boolean indicating whether the **if** clause of the corresponding `pragma omp parallel` is true or false.
int	num_threads	Integer indicating the requested number of threads in the team of the newly created parallel region, or 0 to let the runtime choose the number of threads alone. This attribute may be ignored by the runtime system if the requested number of threads is higher than the number of threads that the runtime can create.

57.24.2.4 struct starpu_omp_task_region_attr

Set of attributes used for creating a new task region.

See also

starpu_omp_task_region()

Data Fields

struct starpu_codelet	cl	starpu_codelet (Codelet And Tasks) to use for the task region explicit task. The codelet must provide a CPU implementation function or an accelerator implementation for offloaded target regions.
starpu_data_handle_t *	handles	Array of zero or more starpu_data_handle_t data handle to be passed to the task region explicit tasks.
void *	cl_arg	Optional pointer to an inline argument to be passed to the region implicit tasks.
size_t	cl_arg_size	Size of the optional inline argument to be passed to the region implicit tasks, or 0 if unused.
unsigned	cl_arg_free	Boolean indicating whether the optional inline argument should be automatically freed (true), or not (false).
int	priority	
int	if_clause	Boolean indicating whether the **if** clause of the corresponding `pragma omp task` is true or false.
int	final_clause	Boolean indicating whether the **final** clause of the corresponding `pragma omp task` is true or false.
int	untied_clause	Boolean indicating whether the **untied** clause of the corresponding `pragma omp task` is true or false.
int	mergeable_clause	Boolean indicating whether the **mergeable** clause of the corresponding `pragma omp task` is true or false.
int	is_loop	taskloop attribute
int	nogroup_clause	
int	collapse	
int	num_tasks	
unsigned long long	nb_iterations	
unsigned long long	grainsize	
unsigned long long	begin_i	
unsigned long long	end_i	
unsigned long long	chunk	

57.24.3 Macro Definition Documentation

57.24.3.1 STARPU_OPENMP

`#define STARPU_OPENMP`

Defined when StarPU has been installed with OpenMP Runtime support. It should be used in your code to detect the availability of the runtime support for OpenMP.

57.24.4 Enumeration Type Documentation

57.24.4.1 starpu_omp_sched_value

`enum starpu_omp_sched_value`

Set of constants for selecting the for loop iteration scheduling algorithm () as defined by the OpenMP specification.

See also

starpu_omp_for()
starpu_omp_for_inline_first()
starpu_omp_for_inline_next()
starpu_omp_for_alt()
starpu_omp_for_inline_first_alt()
starpu_omp_for_inline_next_alt()

Enumerator

starpu_omp_sched_undefined	Undefined iteration scheduling algorithm.
starpu_omp_sched_static	**Static** iteration scheduling algorithm.
starpu_omp_sched_dynamic	**Dynamic** iteration scheduling algorithm.
starpu_omp_sched_guided	**Guided** iteration scheduling algorithm.
starpu_omp_sched_auto	**Automatically** chosen iteration scheduling algorithm.
starpu_omp_sched_runtime	Choice of iteration scheduling algorithm deferred at **runtime**.

57.24.4.2 starpu_omp_proc_bind_value

`enum starpu_omp_proc_bind_value`

Set of constants for selecting the processor binding method, as defined in the OpenMP specification.

See also

starpu_omp_get_proc_bind()

Enumerator

starpu_omp_proc_bind_undefined	Undefined processor binding method.
starpu_omp_proc_bind_false	Team threads may be moved between places at any time.
starpu_omp_proc_bind_true	Team threads may not be moved between places.
starpu_omp_proc_bind_master	Assign every thread in the team to the same place as the **master** thread.
starpu_omp_proc_bind_close	Assign every thread in the team to a place **close** to the parent thread.
starpu_omp_proc_bind_spread	Assign team threads as a sparse distribution over the selected places.

57.24.5 Function Documentation

57.24.5.1 starpu_omp_init()

```
int starpu_omp_init (
            void )
```

Initialize StarPU and its OpenMP Runtime support. See Initialization and Shutdown for more details.

57.24.5.2 starpu_omp_shutdown()

```
void starpu_omp_shutdown (
            void )
```

Shutdown StarPU and its OpenMP Runtime support. See Initialization and Shutdown for more details.

57.24.5.3 starpu_omp_parallel_region()

```
void starpu_omp_parallel_region (
            const struct starpu_omp_parallel_region_attr * attr )
```

Generate and launch an OpenMP parallel region and return after its completion. `attr` specifies the attributes for the generated parallel region. If this function is called from inside another, generating, parallel region, the generated parallel region is nested within the generating parallel region.
This function can be used to implement `#pragma omp parallel`. See Parallel Regions for more details.

57.24.5.4 starpu_omp_master()

```
void starpu_omp_master (
            void(*)(void *arg) f,
            void * arg )
```

Execute a function only on the master thread of the OpenMP parallel region it is called from. When called from a thread that is not the master of the parallel region it is called from, this function does nothing. `f` is the function to be called. `arg` is an argument passed to function `f`.
This function can be used to implement `#pragma omp master`. See Single for more details.

57.24.5.5 starpu_omp_master_inline()

```
int starpu_omp_master_inline (
            void )
```

Determine whether the calling thread is the master of the OpenMP parallel region it is called from or not.
This function can be used to implement `#pragma omp master` without code outlining.

Returns

`!0` if called by the region's master thread.

`0` if not called by the region's master thread. See Single for more details.

57.24.5.6 starpu_omp_barrier()

```
void starpu_omp_barrier (
            void )
```

Wait until each participating thread of the innermost OpenMP parallel region has reached the barrier and each explicit OpenMP task bound to this region has completed its execution.
This function can be used to implement `#pragma omp barrier`. See Barriers for more details.

57.24.5.7 starpu_omp_critical()

```
void starpu_omp_critical (
            void(*)(void *arg) f,
            void * arg,
            const char * name )
```

Wait until no other thread is executing within the context of the selected critical section, then proceeds to the exclusive execution of a function within the critical section. `f` is the function to be executed in the critical section. `arg` is an argument passed to function `f`. `name` is the name of the selected critical section. If `name == NULL`, the selected critical section is the unique anonymous critical section.
This function can be used to implement `#pragma omp critical`.
See Critical Sections for more details.

57.24.5.8 starpu_omp_critical_inline_begin()

```
void starpu_omp_critical_inline_begin (
            const char * name )
```

Wait until execution can proceed exclusively within the context of the selected critical section. `name` is the name of the selected critical section. If `name == NULL`, the selected critical section is the unique anonymous critical section.
This function together with starpu_omp_critical_inline_end can be used to implement `#pragma omp critical` without code outlining.
See Critical Sections for more details.

57.24.5.9 starpu_omp_critical_inline_end()

```
void starpu_omp_critical_inline_end (
            const char * name )
```

End the exclusive execution within the context of the selected critical section. `name` is the name of the selected critical section. If `name==NULL`, the selected critical section is the unique anonymous critical section.
This function together with starpu_omp_critical_inline_begin can be used to implement `#pragma omp critical` without code outlining.
See Critical Sections for more details.

57.24.5.10 starpu_omp_single()

```
void starpu_omp_single (
            void(*)(void *arg) f,
            void * arg,
            int nowait )
```

Ensure that a single participating thread of the innermost OpenMP parallel region executes a function. `f` is the function to be executed by a single thread. `arg` is an argument passed to function `f`. `nowait` is a flag indicating whether an implicit barrier is requested after the single section (`nowait==0`) or not (`nowait==!0`).
This function can be used to implement `#pragma omp single`. See Single for more details.

57.24.5.11 starpu_omp_single_inline()

```
int starpu_omp_single_inline (
            void )
```

Decide whether the current thread is elected to run the following single section among the participating threads of the innermost OpenMP parallel region.
This function can be used to implement `#pragma omp single` without code outlining.

Returns

`!0` if the calling thread has won the election.

`0` if the calling thread has lost the election. See Single for more details.

57.24.5.12 starpu_omp_single_copyprivate()

```
void starpu_omp_single_copyprivate (
            void(*)(void *arg, void *data, unsigned long long data_size) f,
            void * arg,
            void * data,
            unsigned long long data_size )
```

Execute `f` on a single task of the current parallel region task, and then broadcast the contents of the memory block pointed by the copyprivate pointer `data` and of size `data_size` to the corresponding `data` pointed memory blocks of all the other participating region tasks. This function can be used to implement `#pragma omp single` with a copyprivate clause.

See also

starpu_omp_single_copyprivate_inline

starpu_omp_single_copyprivate_inline_begin

starpu_omp_single_copyprivate_inline_end

See Single for more details.

57.24.5.13 starpu_omp_single_copyprivate_inline_begin()

```
void * starpu_omp_single_copyprivate_inline_begin (
            void * data )
```

Elect one task among the tasks of the current parallel region task to execute the following single section, and then broadcast the copyprivate pointer `data` to all the other participating region tasks. This function can be used to implement `#pragma omp single` with a copyprivate clause without code outlining.

See also

starpu_omp_single_copyprivate_inline

starpu_omp_single_copyprivate_inline_end

See Single for more details.

57.24.5.14 starpu_omp_single_copyprivate_inline_end()

```
void starpu_omp_single_copyprivate_inline_end (
            void )
```

Complete the execution of a single section and return the broadcasted copyprivate pointer for tasks that lost the election and `NULL` for the task that won the election. This function can be used to implement `#pragma omp single` with a copyprivate clause without code outlining.
Return the copyprivate pointer for tasks that lost the election and therefore did not execute the code of the single section. Return `NULL` for the task that won the election and executed the code of the single section.

See also

starpu_omp_single_copyprivate_inline

starpu_omp_single_copyprivate_inline_begin

See Single for more details.

57.24.5.15 starpu_omp_for()

```
void starpu_omp_for (
            void(*)(unsigned long long _first_i, unsigned long long _nb_i, void *arg) f,
            void * arg,
            unsigned long long nb_iterations,
            unsigned long long chunk,
            int schedule,
            int ordered,
            int nowait )
```

Execute a parallel loop together with the other threads participating to the innermost parallel region. `f` is the function to be executed iteratively. `arg` is an argument passed to function `f`. `nb_iterations` is the number of iterations to be performed by the parallel loop. `chunk` is the number of consecutive iterations that should be affected to the same thread when scheduling the loop workshares, it follows the semantics of the `modifier` argument in OpenMP `#pragma omp for` specification. `schedule` is the scheduling mode according to the OpenMP specification. `ordered` is a flag indicating whether the loop region may contain an ordered section (`ordered==!0`) or not (`ordered==0`). `nowait` is a flag indicating whether an implicit barrier is requested after the for section (`nowait==0`) or not (`nowait==!0`).
The function `f` will be called with arguments `_first_i`, the first iteration to perform, `_nb_i`, the number of consecutive iterations to perform before returning, `arg`, the free `arg` argument.
This function can be used to implement `#pragma omp for`. See Parallel For for more details.

57.24.5.16 starpu_omp_for_inline_first()

```
int starpu_omp_for_inline_first (
            unsigned long long nb_iterations,
            unsigned long long chunk,
            int schedule,
            int ordered,
            unsigned long long * _first_i,
            unsigned long long * _nb_i )
```

Decide whether the current thread should start to execute a parallel loop section. See starpu_omp_for for the argument description.
This function together with starpu_omp_for_inline_next can be used to implement `#pragma omp for` without code outlining.

Returns

`!0` if the calling thread participates to the loop region and should execute a first chunk of iterations. In that case, `*_first_i` will be set to the first iteration of the chunk to perform and `*_nb_i` will be set to the number of iterations of the chunk to perform.

`0` if the calling thread does not participate to the loop region because all the available iterations have been affected to the other threads of the parallel region.

See also

starpu_omp_for

See Parallel For for more details.

57.24.5.17 starpu_omp_for_inline_next()

```
int starpu_omp_for_inline_next (
            unsigned long long nb_iterations,
            unsigned long long chunk,
            int schedule,
            int ordered,
            unsigned long long * _first_i,
            unsigned long long * _nb_i )
```

Decide whether the current thread should continue to execute a parallel loop section. See starpu_omp_for for the argument description.
This function together with starpu_omp_for_inline_first can be used to implement `#pragma omp for` without code outlining.

Returns

`!0` if the calling thread should execute a next chunk of iterations. In that case, `*_first_i` will be set to the first iteration of the chunk to perform and `*_nb_i` will be set to the number of iterations of the chunk to perform.

`0` if the calling thread does not participate anymore to the loop region because all the available iterations have been affected to the other threads of the parallel region.

See also

starpu_omp_for

See Parallel For for more details.

57.24.5.18 starpu_omp_for_alt()

```
void starpu_omp_for_alt (
            void(*)(unsigned long long _begin_i, unsigned long long _end_i, void *arg) f,
            void * arg,
            unsigned long long nb_iterations,
            unsigned long long chunk,
            int schedule,
            int ordered,
            int nowait )
```

Alternative implementation of a parallel loop. Differ from starpu_omp_for in the expected arguments of the loop function `f`.

The function `f` will be called with arguments `_begin_i`, the first iteration to perform, `_end_i`, the first iteration not to perform before returning, `arg`, the free `arg` argument.
This function can be used to implement `#pragma omp for`.

See also

starpu_omp_for

See Parallel For for more details.

57.24.5.19 starpu_omp_for_inline_first_alt()

```
int starpu_omp_for_inline_first_alt (
            unsigned long long nb_iterations,
            unsigned long long chunk,
            int schedule,
            int ordered,
            unsigned long long * _begin_i,
            unsigned long long * _end_i )
```

Inline version of the alternative implementation of a parallel loop.
This function together with starpu_omp_for_inline_next_alt can be used to implement `#pragma omp for` without code outlining.

See also

starpu_omp_for
starpu_omp_for_alt
starpu_omp_for_inline_first

See Parallel For for more details.

57.24.5.20 starpu_omp_for_inline_next_alt()

```
int starpu_omp_for_inline_next_alt (
            unsigned long long nb_iterations,
            unsigned long long chunk,
            int schedule,
            int ordered,
            unsigned long long * _begin_i,
            unsigned long long * _end_i )
```

Inline version of the alternative implementation of a parallel loop.
This function together with starpu_omp_for_inline_first_alt can be used to implement `#pragma omp for` without code outlining.

See also

starpu_omp_for
starpu_omp_for_alt
starpu_omp_for_inline_next

See Parallel For for more details.

57.24.5.21 starpu_omp_ordered()

```
void starpu_omp_ordered (
            void(*)(void *arg) f,
            void * arg )
```

Ensure that a function is sequentially executed once for each iteration in order within a parallel loop, by the thread that own the iteration. `f` is the function to be executed by the thread that own the current iteration. `arg` is an argument passed to function `f`.
This function can be used to implement `#pragma omp ordered`.
See Parallel For for more details.

57.24.5.22 starpu_omp_ordered_inline_begin()

```
void starpu_omp_ordered_inline_begin (
            void )
```

Wait until all the iterations of a parallel loop below the iteration owned by the current thread have been executed.
This function together with starpu_omp_ordered_inline_end can be used to implement `#pragma omp ordered` without code code outlining.
See Parallel For for more details.

57.24.5.23 starpu_omp_ordered_inline_end()

```
void starpu_omp_ordered_inline_end (
            void )
```

Notify that the ordered section for the current iteration has been completed.
This function together with starpu_omp_ordered_inline_begin can be used to implement `#pragma omp ordered` without code code outlining.
See Parallel For for more details.

57.24.5.24 starpu_omp_sections()

```
void starpu_omp_sections (
            unsigned long long nb_sections,
            void(**)(void *arg) section_f,
            void ** section_arg,
            int nowait )
```

Ensure that each function of a given array of functions is executed by one and only one thread. `nb_sections` is the number of functions in the array `section_f`. `section_f` is the array of functions to be executed as sections. `section_arg` is an array of arguments to be passed to the corresponding function. `nowait` is a flag indicating whether an implicit barrier is requested after the execution of all the sections (`nowait==0`) or not (`nowait==!0`).
This function can be used to implement `#pragma omp sections` and `#pragma omp section`.
See Sections for more details.

57.24.5.25 starpu_omp_sections_combined()

```
void starpu_omp_sections_combined (
            unsigned long long nb_sections,
            void(*)(unsigned long long section_num, void *arg) section_f,
            void * section_arg,
            int nowait )
```

Alternative implementation of sections. Differ from starpu_omp_sections in that all the sections are combined within a single function in this version. `section_f` is the function implementing the combined sections.
The function `section_f` will be called with arguments `section_num`, the section number to be executed, `arg`, the entry of `section_arg` corresponding to this section.
This function can be used to implement `#pragma omp sections` and `#pragma omp section`.

See also

starpu_omp_sections

See Sections for more details.

57.24.5.26 starpu_omp_task_region()

```
void starpu_omp_task_region (
            const struct starpu_omp_task_region_attr * attr )
```

Generate an explicit child task. The execution of the generated task is asynchronous with respect to the calling code unless specified otherwise. `attr` specifies the attributes for the generated task region.
This function can be used to implement `#pragma omp task`.
See Explicit Tasks for more details.

57.24.5.27 starpu_omp_taskwait()

```
void starpu_omp_taskwait (
            void )
```

Wait for the completion of the tasks generated by the current task. This function does not wait for the descendants of the tasks generated by the current task.
This function can be used to implement `#pragma omp taskwait`.
See TaskWait and TaskGroup for more details.

57.24.5.28 starpu_omp_taskgroup()

```
void starpu_omp_taskgroup (
            void(*)(void *arg) f,
            void * arg )
```

Launch a function and wait for the completion of every descendant task generated during the execution of the function.
This function can be used to implement `#pragma omp taskgroup`.

See also

starpu_omp_taskgroup_inline_begin

starpu_omp_taskgroup_inline_end

See TaskWait and TaskGroup for more details.

57.24.5.29 starpu_omp_taskgroup_inline_begin()

```
void starpu_omp_taskgroup_inline_begin (
            void )
```

Launch a function and gets ready to wait for the completion of every descendant task generated during the dynamic scope of the taskgroup.
This function can be used to implement `#pragma omp taskgroup` without code outlining.

See also

starpu_omp_taskgroup

starpu_omp_taskgroup_inline_end

See TaskWait and TaskGroup for more details.

57.24.5.30 starpu_omp_taskgroup_inline_end()

```
void starpu_omp_taskgroup_inline_end (
            void )
```

Wait for the completion of every descendant task generated during the dynamic scope of the taskgroup.
This function can be used to implement `#pragma omp taskgroup` without code outlining.

See also

starpu_omp_taskgroup

starpu_omp_taskgroup_inline_begin

See TaskWait and TaskGroup for more details.

57.24.5.31 starpu_omp_set_num_threads()

```
void starpu_omp_set_num_threads (
            int threads )
```

Set ICVS nthreads_var for the parallel regions to be created with the current region.
Note: The StarPU OpenMP runtime support currently ignores this setting for nested parallel regions.

See also

starpu_omp_get_num_threads
starpu_omp_get_thread_num
starpu_omp_get_max_threads
starpu_omp_get_num_procs

See OpenMP Standard Functions in StarPU for more details.

57.24.5.32 starpu_omp_get_num_threads()

```
int starpu_omp_get_num_threads (
              void )
```

Return the number of threads of the current region.

Returns

the number of threads of the current region.

See also

starpu_omp_set_num_threads
starpu_omp_get_thread_num
starpu_omp_get_max_threads
starpu_omp_get_num_procs

See OpenMP Standard Functions in StarPU for more details.

57.24.5.33 starpu_omp_get_thread_num()

```
int starpu_omp_get_thread_num (
              void )
```

Return the rank of the current thread among the threads of the current region.

Returns

the rank of the current thread in the current region.

See also

starpu_omp_set_num_threads
starpu_omp_get_num_threads
starpu_omp_get_max_threads
starpu_omp_get_num_procs

See OpenMP Standard Functions in StarPU for more details.

57.24.5.34 starpu_omp_get_max_threads()

```
int starpu_omp_get_max_threads (
              void )
```

Return the maximum number of threads that can be used to create a region from the current region.

Returns

the maximum number of threads that can be used to create a region from the current region.

See also

starpu_omp_set_num_threads
starpu_omp_get_num_threads
starpu_omp_get_thread_num
starpu_omp_get_num_procs

See OpenMP Standard Functions in StarPU for more details.

57.24.5.35 starpu_omp_get_num_procs()

```
int starpu_omp_get_num_procs (
            void )
```

Return the number of StarPU CPU workers.

Returns

the number of StarPU CPU workers.

See also

starpu_omp_set_num_threads

starpu_omp_get_num_threads

starpu_omp_get_thread_num

starpu_omp_get_max_threads

See OpenMP Standard Functions in StarPU for more details.

57.24.5.36 starpu_omp_in_parallel()

```
int starpu_omp_in_parallel (
            void )
```

Return whether it is called from the scope of a parallel region or not.

Returns

`!0` if called from a parallel region scope.

`0` otherwise.

See OpenMP Standard Functions in StarPU for more details.

57.24.5.37 starpu_omp_set_dynamic()

```
void starpu_omp_set_dynamic (
            int dynamic_threads )
```

Enable (1) or disable (0) dynamically adjusting the number of parallel threads.
Note: The StarPU OpenMP runtime support currently ignores the argument of this function.

See also

starpu_omp_get_dynamic

See OpenMP Standard Functions in StarPU for more details.

57.24.5.38 starpu_omp_get_dynamic()

```
int starpu_omp_get_dynamic (
            void )
```

Return the state of dynamic thread number adjustment.

Returns

`!0` if dynamic thread number adjustment is enabled.

`0` otherwise.

See also

starpu_omp_set_dynamic

See OpenMP Standard Functions in StarPU for more details.

57.24.5.39 starpu_omp_set_nested()

```
void starpu_omp_set_nested (
            int nested )
```

Enable (1) or disable (0) nested parallel regions.
Note: The StarPU OpenMP runtime support currently ignores the argument of this function.

See also

- starpu_omp_get_nested
- starpu_omp_get_max_active_levels
- starpu_omp_set_max_active_levels
- starpu_omp_get_level
- starpu_omp_get_active_level

See OpenMP Standard Functions in StarPU for more details.

57.24.5.40 starpu_omp_get_nested()

```
int starpu_omp_get_nested (
            void  )
```

Return whether nested parallel sections are enabled or not.

Returns

`!0` if nested parallel sections are enabled.

`0` otherwise.

See also

- starpu_omp_set_nested
- starpu_omp_get_max_active_levels
- starpu_omp_set_max_active_levels
- starpu_omp_get_level
- starpu_omp_get_active_level

See OpenMP Standard Functions in StarPU for more details.

57.24.5.41 starpu_omp_get_cancellation()

```
int starpu_omp_get_cancellation (
            void  )
```

Return the state of the cancel ICVS var.
See OpenMP Standard Functions in StarPU for more details.

57.24.5.42 starpu_omp_set_schedule()

```
void starpu_omp_set_schedule (
            enum starpu_omp_sched_value kind,
            int modifier )
```

Set the default scheduling kind for upcoming loops within the current parallel section. `kind` is the scheduler kind, `modifier` complements the scheduler kind with information such as the chunk size, in accordance with the OpenMP specification.

See also

- starpu_omp_get_schedule

See Parallel For for more details.

57.24.5.43 starpu_omp_get_schedule()

```
void starpu_omp_get_schedule (
            enum starpu_omp_sched_value * kind,
            int * modifier )
```

Return the kind and the modifier of the current default loop scheduler.

See also

starpu_omp_set_schedule

See OpenMP Standard Functions in StarPU for more details.

57.24.5.44 starpu_omp_get_thread_limit()

```
int starpu_omp_get_thread_limit (
            void )
```

Return the number of StarPU CPU workers.

Returns

the number of StarPU CPU workers.

See OpenMP Standard Functions in StarPU for more details.

57.24.5.45 starpu_omp_set_max_active_levels()

```
void starpu_omp_set_max_active_levels (
            int max_levels )
```

Set the maximum number of allowed active parallel section levels.
Note: The StarPU OpenMP runtime support currently ignores the argument of this function and assume `max_levels` equals `1` instead.

See also

starpu_omp_set_nested

starpu_omp_get_nested

starpu_omp_get_max_active_levels

starpu_omp_get_level

starpu_omp_get_active_level

See OpenMP Standard Functions in StarPU for more details.

57.24.5.46 starpu_omp_get_max_active_levels()

```
int starpu_omp_get_max_active_levels (
            void )
```

Return the current maximum number of allowed active parallel section levels

Returns

the current maximum number of allowed active parallel section levels.

See also

starpu_omp_set_nested

starpu_omp_get_nested

starpu_omp_set_max_active_levels

starpu_omp_get_level

starpu_omp_get_active_level

See OpenMP Standard Functions in StarPU for more details.

57.24.5.47 starpu_omp_get_level()

```
int starpu_omp_get_level (
            void )
```

Return the nesting level of the current parallel section.

Returns

the nesting level of the current parallel section.

See also

starpu_omp_set_nested

starpu_omp_get_nested

starpu_omp_get_max_active_levels

starpu_omp_set_max_active_levels

starpu_omp_get_active_level

See OpenMP Standard Functions in StarPU for more details.

57.24.5.48 starpu_omp_get_ancestor_thread_num()

```
int starpu_omp_get_ancestor_thread_num (
            int level )
```

Return the number of the ancestor of the current parallel section.

Returns

the number of the ancestor of the current parallel section.

See OpenMP Standard Functions in StarPU for more details.

57.24.5.49 starpu_omp_get_team_size()

```
int starpu_omp_get_team_size (
            int level )
```

Return the size of the team of the current parallel section.

Returns

the size of the team of the current parallel section.

See OpenMP Standard Functions in StarPU for more details.

57.24.5.50 starpu_omp_get_active_level()

```
int starpu_omp_get_active_level (
            void )
```

Return the nestinglevel of the current innermost active parallel section.

Returns

the nestinglevel of the current innermost active parallel section.

See also

starpu_omp_set_nested

starpu_omp_get_nested

starpu_omp_get_max_active_levels

starpu_omp_set_max_active_levels

starpu_omp_get_level

See OpenMP Standard Functions in StarPU for more details.

57.24.5.51 starpu_omp_in_final()

```
int starpu_omp_in_final (
            void  )
```

Check whether the current task is final or not.

Returns

`!0` if called from a final task.

`0` otherwise.

See OpenMP Standard Functions in StarPU for more details.

57.24.5.52 starpu_omp_get_proc_bind()

```
enum starpu_omp_proc_bind_value starpu_omp_get_proc_bind (
            void  )
```

Return the proc_bind setting of the current parallel region.

Returns

the proc_bind setting of the current parallel region.

See OpenMP Standard Functions in StarPU for more details.

57.24.5.53 starpu_omp_get_num_places()

```
int starpu_omp_get_num_places (
            void  )
```

Return the number of places available to the execution environment in the place list.

Returns

the number of places available to the execution environment in the place list.

See OpenMP Standard Functions in StarPU for more details.

57.24.5.54 starpu_omp_get_place_num_procs()

```
int starpu_omp_get_place_num_procs (
            int place_num )
```

Return the number of processors available to the execution environment in the specified place.

Returns

the number of processors available to the execution environment in the specified place.

See OpenMP Standard Functions in StarPU for more details.

57.24.5.55 starpu_omp_get_place_proc_ids()

```
void starpu_omp_get_place_proc_ids (
            int place_num,
            int * ids )
```

Return the numerical identifiers of the processors available to the execution environment in the specified place.
See OpenMP Standard Functions in StarPU for more details.

57.24.5.56 starpu_omp_get_place_num()

```
int starpu_omp_get_place_num (
            void  )
```

Return the place number of the place to which the encountering thread is bound.

Returns

the place number of the place to which the encountering thread is bound.

See OpenMP Standard Functions in StarPU for more details.

57.24.5.57 starpu_omp_get_partition_num_places()

```
int starpu_omp_get_partition_num_places (
            void )
```

Return the number of places in the place partition of the innermost implicit task.

Returns

the number of places in the place partition of the innermost implicit task.

See OpenMP Standard Functions in StarPU for more details.

57.24.5.58 starpu_omp_get_partition_place_nums()

```
void starpu_omp_get_partition_place_nums (
            int * place_nums )
```

Return the list of place numbers corresponding to the places in the place-partition-var ICV of the innermost implicit task.
See OpenMP Standard Functions in StarPU for more details.

57.24.5.59 starpu_omp_set_default_device()

```
void starpu_omp_set_default_device (
            int device_num )
```

Set the number of the device to use as default.
Note: The StarPU OpenMP runtime support currently ignores the argument of this function.

See also

starpu_omp_get_default_device

starpu_omp_is_initial_device

See OpenMP Standard Functions in StarPU for more details.

57.24.5.60 starpu_omp_get_default_device()

```
int starpu_omp_get_default_device (
            void )
```

Return the number of the device used as default.

Returns

the number of the device used as default.

See also

starpu_omp_set_default_device

starpu_omp_is_initial_device

See OpenMP Standard Functions in StarPU for more details.

57.24.5.61 starpu_omp_get_num_devices()

```
int starpu_omp_get_num_devices (
            void )
```

Return the number of the devices.

Returns

the number of the devices.

See OpenMP Standard Functions in StarPU for more details.

57.24.5.62 starpu_omp_get_num_teams()

```
int starpu_omp_get_num_teams (
             void   )
```

Return the number of teams in the current teams region.

Returns

the number of teams in the current teams region.

See also

starpu_omp_get_num_teams

See OpenMP Standard Functions in StarPU for more details.

57.24.5.63 starpu_omp_get_team_num()

```
int starpu_omp_get_team_num (
             void   )
```

Return the team number of the calling thread.

Returns

the team number of the calling thread.

See also

starpu_omp_get_num_teams

See OpenMP Standard Functions in StarPU for more details.

57.24.5.64 starpu_omp_is_initial_device()

```
int starpu_omp_is_initial_device (
             void   )
```

Check whether the current device is the initial device or not.
See OpenMP Standard Functions in StarPU for more details.

57.24.5.65 starpu_omp_get_initial_device()

```
int starpu_omp_get_initial_device (
             void   )
```

Return a device number that represents the host device.

Returns

a device number that represents the host device.

See OpenMP Standard Functions in StarPU for more details.

57.24.5.66 starpu_omp_get_max_task_priority()

```
int starpu_omp_get_max_task_priority (
             void   )
```

Return the maximum value that can be specified in the priority clause.

Returns

`!0` if called from the host device.

`0` otherwise.

See also

starpu_omp_set_default_device

starpu_omp_get_default_device

See OpenMP Standard Functions in StarPU for more details.

57.24.5.67 starpu_omp_init_lock()

```
void starpu_omp_init_lock (
            starpu_omp_lock_t * lock )
```

Initialize an opaque lock object.

See also

- starpu_omp_destroy_lock
- starpu_omp_set_lock
- starpu_omp_unset_lock
- starpu_omp_test_lock

See Simple Locks for more details.

57.24.5.68 starpu_omp_destroy_lock()

```
void starpu_omp_destroy_lock (
            starpu_omp_lock_t * lock )
```

Destroy an opaque lock object.

See also

- starpu_omp_init_lock
- starpu_omp_set_lock
- starpu_omp_unset_lock
- starpu_omp_test_lock

See Simple Locks for more details.

57.24.5.69 starpu_omp_set_lock()

```
void starpu_omp_set_lock (
            starpu_omp_lock_t * lock )
```

Lock an opaque lock object. If the lock is already locked, the function will block until it succeeds in exclusively acquiring the lock.

See also

- starpu_omp_init_lock
- starpu_omp_destroy_lock
- starpu_omp_unset_lock
- starpu_omp_test_lock

See Simple Locks for more details.

57.24.5.70 starpu_omp_unset_lock()

```
void starpu_omp_unset_lock (
            starpu_omp_lock_t * lock )
```

Unlock a previously locked lock object. The behaviour of this function is unspecified if it is called on an unlocked lock object.

See also

- starpu_omp_init_lock
- starpu_omp_destroy_lock
- starpu_omp_set_lock
- starpu_omp_test_lock

See Simple Locks for more details.

57.24.5.71 starpu_omp_test_lock()

```
int starpu_omp_test_lock (
            starpu_omp_lock_t * lock )
```

Unblockingly attempt to lock a lock object and return whether it succeeded or not.

Returns

`!0` if the function succeeded in acquiring the lock.

`0` if the lock was already locked.

See also

starpu_omp_init_lock

starpu_omp_destroy_lock

starpu_omp_set_lock

starpu_omp_unset_lock

See Simple Locks for more details.

57.24.5.72 starpu_omp_init_nest_lock()

```
void starpu_omp_init_nest_lock (
            starpu_omp_nest_lock_t * lock )
```

Initialize an opaque lock object supporting nested locking operations.

See also

starpu_omp_destroy_nest_lock

starpu_omp_set_nest_lock

starpu_omp_unset_nest_lock

starpu_omp_test_nest_lock

See Nestable Locks for more details.

57.24.5.73 starpu_omp_destroy_nest_lock()

```
void starpu_omp_destroy_nest_lock (
            starpu_omp_nest_lock_t * lock )
```

Destroy an opaque lock object supporting nested locking operations.

See also

starpu_omp_init_nest_lock

starpu_omp_set_nest_lock

starpu_omp_unset_nest_lock

starpu_omp_test_nest_lock

See Nestable Locks for more details.

57.24.5.74 starpu_omp_set_nest_lock()

```
void starpu_omp_set_nest_lock (
            starpu_omp_nest_lock_t * lock )
```

Lock an opaque lock object supporting nested locking operations. If the lock is already locked by another task, the function will block until it succeeds in exclusively acquiring the lock. If the lock is already taken by the current task, the function will increase the nested locking level of the lock object.

See also

- starpu_omp_init_nest_lock
- starpu_omp_destroy_nest_lock
- starpu_omp_unset_nest_lock
- starpu_omp_test_nest_lock

See Nestable Locks for more details.

57.24.5.75 starpu_omp_unset_nest_lock()

```
void starpu_omp_unset_nest_lock (
            starpu_omp_nest_lock_t * lock )
```

Unlock a previously locked lock object supporting nested locking operations. If the lock has been locked multiple times in nested fashion, the nested locking level is decreased and the lock remains locked. Otherwise, if the lock has only been locked once, it becomes unlocked. The behaviour of this function is unspecified if it is called on an unlocked lock object. The behaviour of this function is unspecified if it is called from a different task than the one that locked the lock object.

See also

- starpu_omp_init_nest_lock
- starpu_omp_destroy_nest_lock
- starpu_omp_set_nest_lock
- starpu_omp_test_nest_lock

See Nestable Locks for more details.

57.24.5.76 starpu_omp_test_nest_lock()

```
int starpu_omp_test_nest_lock (
            starpu_omp_nest_lock_t * lock )
```

Unblocking attempt to lock an opaque lock object supporting nested locking operations and returns whether it succeeded or not. If the lock is already locked by another task, the function will return without having acquired the lock. If the lock is already taken by the current task, the function will increase the nested locking level of the lock object.

Returns

`!0` if the function succeeded in acquiring the lock.

`0` if the lock was already locked.

See also

- starpu_omp_init_nest_lock
- starpu_omp_destroy_nest_lock
- starpu_omp_set_nest_lock
- starpu_omp_unset_nest_lock

See Nestable Locks for more details.

57.24.5.77 starpu_omp_atomic_fallback_inline_begin()

```
void starpu_omp_atomic_fallback_inline_begin (
            void  )
```

Implement the entry point of a fallback global atomic region. Block until it succeeds in acquiring exclusive access to the global atomic region.

See also

- starpu_omp_atomic_fallback_inline_end

57.24.5.78 starpu_omp_atomic_fallback_inline_end()

```
void starpu_omp_atomic_fallback_inline_end (
            void )
```

Implement the exit point of a fallback global atomic region. Release the exclusive access to the global atomic region.

See also

starpu_omp_atomic_fallback_inline_begin

57.24.5.79 starpu_omp_get_wtime()

```
double starpu_omp_get_wtime (
            void )
```

Return the elapsed wallclock time in seconds.

Returns

the elapsed wallclock time in seconds.

See also

starpu_omp_get_wtick

See OpenMP Standard Functions in StarPU for more details.

57.24.5.80 starpu_omp_get_wtick()

```
double starpu_omp_get_wtick (
            void )
```

Return the precision of the time used by `starpu_omp_get_wtime()`.

Returns

the precision of the time used by `starpu_omp_get_wtime()`.

See also

starpu_omp_get_wtime

See OpenMP Standard Functions in StarPU for more details.

57.24.5.81 starpu_omp_vector_annotate()

```
void starpu_omp_vector_annotate (
            starpu_data_handle_t handle,
            uint32_t slice_base )
```

Enable setting additional vector metadata needed by the OpenMP Runtime Support.
`handle` is vector data handle. `slice_base` is the base of an array slice, expressed in number of vector elements from the array base.

See also

STARPU_VECTOR_GET_SLICE_BASE

57.24.5.82 starpu_omp_get_default_arbiter()

```
struct starpu_arbiter * starpu_omp_get_default_arbiter (
            void )
```

Only use internally by StarPU.

57.24.5.83 starpu_omp_handle_register()

```
void starpu_omp_handle_register (
            starpu_data_handle_t handle )
```

Register a handle for ptr->handle data lookup.

See also

starpu_omp_handle_unregister

starpu_omp_data_lookup

See Data Dependencies for more details.

57.24.5.84 starpu_omp_handle_unregister()

```
void starpu_omp_handle_unregister (
            starpu_data_handle_t handle )
```

Unregister a handle from ptr->handle data lookup.

See also

starpu_omp_handle_register

starpu_omp_data_lookup

See Data Dependencies for more details.

57.24.5.85 starpu_omp_data_lookup()

```
starpu_data_handle_t starpu_omp_data_lookup (
            const void * ptr )
```

Return the handle corresponding to the data pointed to by the `ptr` host pointer.

Returns

the handle or `NULL` if not found.

See Data Dependencies for more details.

57.25 Out Of Core

Data Structures

- struct starpu_disk_ops

Macros

- #define STARPU_DISK_SIZE_MIN

Functions

- void starpu_disk_close (unsigned node, void ∗obj, size_t size)
- void ∗ starpu_disk_open (unsigned node, void ∗pos, size_t size)
- int starpu_disk_register (struct starpu_disk_ops ∗func, void ∗parameter, starpu_ssize_t size)

Variables

- struct starpu_disk_ops starpu_disk_stdio_ops
- struct starpu_disk_ops starpu_disk_hdf5_ops
- struct starpu_disk_ops starpu_disk_unistd_ops
- struct starpu_disk_ops starpu_disk_unistd_o_direct_ops
- struct starpu_disk_ops starpu_disk_leveldb_ops
- int starpu_disk_swap_node

57.25.1 Detailed Description

57.25.2 Data Structure Documentation

57.25.2.1 struct starpu_disk_ops

Set of functions to manipulate data on disk. See Disk functions for more details.

Data Fields

- void ∗(∗ plug)(void ∗parameter, starpu_ssize_t size)
- void(∗ unplug)(void ∗base)
- int(∗ bandwidth)(unsigned node, void ∗base)
- void ∗(∗ alloc)(void ∗base, size_t size)
- void(∗ free)(void ∗base, void ∗obj, size_t size)
- void ∗(∗ open)(void ∗base, void ∗pos, size_t size)
- void(∗ close)(void ∗base, void ∗obj, size_t size)
- int(∗ read)(void ∗base, void ∗obj, void ∗buf, off_t offset, size_t size)
- int(∗ write)(void ∗base, void ∗obj, const void ∗buf, off_t offset, size_t size)
- int(∗ full_read)(void ∗base, void ∗obj, void ∗∗ptr, size_t ∗size, unsigned dst_node)
- int(∗ full_write)(void ∗base, void ∗obj, void ∗ptr, size_t size)
- void ∗(∗ async_write)(void ∗base, void ∗obj, void ∗buf, off_t offset, size_t size)
- void ∗(∗ async_read)(void ∗base, void ∗obj, void ∗buf, off_t offset, size_t size)
- void ∗(∗ async_full_read)(void ∗base, void ∗obj, void ∗∗ptr, size_t ∗size, unsigned dst_node)
- void ∗(∗ async_full_write)(void ∗base, void ∗obj, void ∗ptr, size_t size)
- void ∗(∗ copy)(void ∗base_src, void ∗obj_src, off_t offset_src, void ∗base_dst, void ∗obj_dst, off_t offset_dst, size_t size)
- void(∗ wait_request)(void ∗async_channel)
- int(∗ test_request)(void ∗async_channel)
- void(∗ free_request)(void ∗async_channel)

57.25.2.1.1 Field Documentation

57.25.2.1.1.1 plug `void *(* starpu_disk_ops::plug) (void *parameter, starpu_ssize_t size)`

Connect a disk memory at location `parameter` with size `size`, and return a base as void∗, which will be passed by StarPU to all other methods.

57.25.2.1.1.2 unplug `void(* starpu_disk_ops::unplug) (void *base)`

Disconnect a disk memory `base`.

57.25.2.1.1.3 bandwidth `int(* starpu_disk_ops::bandwidth) (unsigned node, void *base)`

Measure the bandwidth and the latency for the disk `node` and save it. Returns 1 if it could measure it.

57.25.2.1.1.4 alloc `void *(* starpu_disk_ops::alloc) (void *base, size_t size)`

Create a new location for data of size `size`. Return an opaque object pointer.

57.25.2.1.1.5 free `void(* starpu_disk_ops::free) (void *base, void *obj, size_t size)`

Free a data `obj` previously allocated with starpu_disk_ops::alloc.

57.25.2.1.1.6 open `void *(* starpu_disk_ops::open) (void *base, void *pos, size_t size)`

Open an existing location of data, at a specific position `pos` dependent on the backend.

57.25.2.1.1.7 close `void(* starpu_disk_ops::close) (void *base, void *obj, size_t size)`

Close, without deleting it, a location of data `obj`.

57.25.2.1.1.8 read `int(* starpu_disk_ops::read) (void *base, void *obj, void *buf, off_t offset, size_t size)`

Read `size` bytes of data from `obj` in `base`, at offset `offset`, and put into `buf`. Return the actual number of read bytes.

57.25.2.1.1.9 write `int(* starpu_disk_ops::write) (void *base, void *obj, const void *buf, off_t offset, size_t size)`

Write `size` bytes of data to `obj` in `base`, at offset `offset`, from `buf`. Return 0 on success.

57.25.2.1.1.10 full_read `int(* starpu_disk_ops::full_read) (void *base, void *obj, void **ptr, size_t *size, unsigned dst_node)`

Read all data from `obj` of `base`, from offset 0. Returns it in an allocated buffer `ptr`, of size `size`

57.25.2.1.1.11 full_write `int(* starpu_disk_ops::full_write) (void *base, void *obj, void *ptr, size_t size)`

Write data in `ptr` to `obj` of `base`, from offset 0, and truncate `obj` to `size`, so that a `full_read` will get it.

57.25.2.1.1.12 async_write `void *(* starpu_disk_ops::async_write) (void *base, void *obj, void *buf, off_t offset, size_t size)`

Asynchronously write `size` bytes of data to `obj` in `base`, at offset `offset`, from `buf`. Return a void∗ pointer that StarPU will pass to `xxx_request` methods for testing for the completion.

57.25.2.1.1.13 async_read `void *(* starpu_disk_ops::async_read) (void *base, void *obj, void *buf, off_t offset, size_t size)`

Asynchronously read `size` bytes of data from `obj` in `base`, at offset `offset`, and put into `buf`. Return a void∗ pointer that StarPU will pass to `xxx_request` methods for testing for the completion.

57.25.2.1.1.14 async_full_read `void *(* starpu_disk_ops::async_full_read) (void *base, void *obj, void **ptr, size_t *size, unsigned dst_node)`

Read all data from `obj` of `base`, from offset 0. Return it in an allocated buffer `ptr`, of size `size`

57.25.2.1.1.15 async_full_write `void *(* starpu_disk_ops::async_full_write) (void *base, void *obj, void *ptr, size_t size)`

Write data in `ptr` to `obj` of `base`, from offset 0, and truncate `obj` to `size`, so that a starpu_disk_ops::full_read will get it.

57.25.2.1.1.16 copy `void *(* starpu_disk_ops::copy) (void *base_src, void *obj_src, off_t offset_src, void *base_dst, void *obj_dst, off_t offset_dst, size_t size)`

Copy from offset `offset_src` of disk object `obj_src` in `base_src` to offset `offset_dst` of disk object `obj_dst` in `base_dst`. Return a void∗ pointer that StarPU will pass to `xxx_request` methods for testing for the completion.

57.25.2.1.1.17 wait_request `void(* starpu_disk_ops::wait_request) (void *async_channel)`

Wait for completion of request `async_channel` returned by a previous asynchronous read, write or copy.

57.25.2.1.1.18 test_request `int(* starpu_disk_ops::test_request) (void *async_channel)`

Test for completion of request `async_channel` returned by a previous asynchronous read, write or copy. Return 1 on completion, 0 otherwise.

57.25.2.1.1.19 free_request `void(* starpu_disk_ops::free_request) (void *async_channel)`

Free the request allocated by a previous asynchronous read, write or copy.

57.25.3 Macro Definition Documentation

57.25.3.1 STARPU_DISK_SIZE_MIN

```
#define STARPU_DISK_SIZE_MIN
```

Minimum size of a registered disk. The size of a disk is the last parameter of the function starpu_disk_register().

57.25.4 Function Documentation

57.25.4.1 starpu_disk_close()

```
void starpu_disk_close (
             unsigned node,
             void * obj,
             size_t size )
```

Close an existing data opened with starpu_disk_open(). See Introduction for more details.

57.25.4.2 starpu_disk_open()

```
void * starpu_disk_open (
             unsigned node,
             void * pos,
             size_t size )
```

Open an existing file memory in a disk node. `size` is the size of the file. `pos` is the specific position dependent on the backend, given to the `open` method of the disk operations. Return an opaque object pointer. See Introduction for more details.

57.25.4.3 starpu_disk_register()

```
int starpu_disk_register (
             struct starpu_disk_ops * func,
             void * parameter,
             starpu_ssize_t size )
```

Register a disk memory node with a set of functions to manipulate data. The `plug` member of `func` will be passed `parameter`, and return a `base` which will be passed to all `func` methods.
SUCCESS: return the disk node.
FAIL: return an error code.
`size` must be at least STARPU_DISK_SIZE_MIN bytes ! `size` being negative means infinite size.
See Introduction for more details.

57.25.5 Variable Documentation

57.25.5.1 starpu_disk_stdio_ops

`struct starpu_disk_ops starpu_disk_stdio_ops [extern]`
Use the stdio library (fwrite, fread...) to read/write on disk.
Warning: It creates one file per allocation !
Do not support asynchronous transfers.

57.25.5.2 starpu_disk_hdf5_ops

`struct starpu_disk_ops starpu_disk_hdf5_ops [extern]`
Use the HDF5 library.
It doesn't support multiple opening from different processes.
You may only allow one process to write in the HDF5 file.
If HDF5 library is not compiled with –thread-safe you can't open more than one HDF5 file at the same time.

57.25.5.3 starpu_disk_unistd_ops

`struct starpu_disk_ops starpu_disk_unistd_ops [extern]`
Use the unistd library (write, read...) to read/write on disk.
Warning: It creates one file per allocation !

57.25.5.4 starpu_disk_unistd_o_direct_ops

`struct starpu_disk_ops starpu_disk_unistd_o_direct_ops [extern]`
Use the unistd library (write, read...) to read/write on disk with the O_DIRECT flag.
Warning: It creates one file per allocation !
Only available on Linux systems.

57.25.5.5 starpu_disk_leveldb_ops

`struct starpu_disk_ops starpu_disk_leveldb_ops [extern]`
Use the leveldb created by Google. More information at `https://code.google.com/p/leveldb/` Do not support asynchronous transfers.

57.25.5.6 starpu_disk_swap_node

`int starpu_disk_swap_node [extern]`
Contain the node number of the disk swap, if set up through the STARPU_DISK_SWAP variable.

57.26 Parallel Tasks

Functions

- unsigned starpu_combined_worker_get_count (void)
- unsigned starpu_worker_is_combined_worker (int id)
- int starpu_combined_worker_get_id (void)
- int starpu_combined_worker_get_size (void)
- int starpu_combined_worker_get_rank (void)
- int starpu_combined_worker_assign_workerid (int nworkers, int workerid_array[ ])
- int starpu_combined_worker_get_description (int workerid, int ∗worker_size, int ∗∗combined_workerid)
- int starpu_combined_worker_can_execute_task (unsigned workerid, struct starpu_task ∗task, unsigned nimpl)
- void starpu_parallel_task_barrier_init (struct starpu_task ∗task, int workerid)
- void starpu_parallel_task_barrier_init_n (struct starpu_task ∗task, int worker_size)

57.26.1 Detailed Description

57.26.2 Function Documentation

57.26.2.1 starpu_combined_worker_get_count()

```
unsigned starpu_combined_worker_get_count (
            void )
```

Return the number of different combined workers. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.26.2.2 starpu_worker_is_combined_worker()

```
unsigned starpu_worker_is_combined_worker (
            int id )
```

See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.26.2.3 starpu_combined_worker_get_id()

```
int starpu_combined_worker_get_id (
            void )
```

Return the identifier of the current combined worker. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.26.2.4 starpu_combined_worker_get_size()

```
int starpu_combined_worker_get_size (
            void )
```

Return the size of the current combined worker, i.e. the total number of CPUS running the same task in the case of STARPU_SPMD parallel tasks, or the total number of threads that the task is allowed to start in the case of STARPU_FORKJOIN parallel tasks. See Fork-mode Parallel Tasks and SPMD-mode Parallel Tasks for more details.

57.26.2.5 starpu_combined_worker_get_rank()

```
int starpu_combined_worker_get_rank (
            void )
```

Return the rank of the current thread within the combined worker. Can only be used in STARPU_SPMD parallel tasks, to know which part of the task to work on. See SPMD-mode Parallel Tasks for more details.

57.26.2.6 starpu_combined_worker_assign_workerid()

```
int starpu_combined_worker_assign_workerid (
            int nworkers,
            int workerid_array[ ] )
```

Register a new combined worker and get its identifier. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.26.2.7 starpu_combined_worker_get_description()

```
int starpu_combined_worker_get_description (
            int workerid,
            int * worker_size,
            int ** combined_workerid )
```

Get the description of a combined worker. See Helper functions for defining a scheduling policy (Basic or modular) for more details.
`workerid` is the requested combined worker id, `worker_size` returns the number of workers in the combined worker, `combined_workerid` returns the list for worker ids in the combined worker.

57.26.2.8 starpu_combined_worker_can_execute_task()

```
int starpu_combined_worker_can_execute_task (
            unsigned workerid,
            struct starpu_task * task,
            unsigned nimpl )
```

Variant of starpu_worker_can_execute_task() compatible with combined workers. See Defining A New Basic Scheduling Policy for more details.

57.26.2.9 starpu_parallel_task_barrier_init()

```
void starpu_parallel_task_barrier_init (
            struct starpu_task * task,
            int workerid )
```

Initialise the barrier for the parallel task, and dispatch the task between the different workers of the given combined worker. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.26.2.10 starpu_parallel_task_barrier_init_n()

```
void starpu_parallel_task_barrier_init_n (
            struct starpu_task * task,
            int worker_size )
```

Initialise the barrier for the parallel task, to be pushed to `worker_size` workers (without having to explicit a given combined worker). See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.27 Parallel Workers

Macros

- #define STARPU_PARALLEL_WORKER_MIN_NB
- #define STARPU_PARALLEL_WORKER_MAX_NB
- #define STARPU_PARALLEL_WORKER_NB
- #define STARPU_PARALLEL_WORKER_PREFERE_MIN
- #define STARPU_PARALLEL_WORKER_KEEP_HOMOGENEOUS
- #define STARPU_PARALLEL_WORKER_POLICY_NAME
- #define STARPU_PARALLEL_WORKER_POLICY_STRUCT
- #define STARPU_PARALLEL_WORKER_CREATE_FUNC
- #define STARPU_PARALLEL_WORKER_CREATE_FUNC_ARG
- #define STARPU_PARALLEL_WORKER_TYPE
- #define STARPU_PARALLEL_WORKER_AWAKE_WORKERS
- #define STARPU_PARALLEL_WORKER_PARTITION_ONE
- #define STARPU_PARALLEL_WORKER_NEW
- #define STARPU_PARALLEL_WORKER_NCORES
- #define **starpu_parallel_worker_intel_openmp_mkl_prologue**
- #define STARPU_CLUSTER_MIN_NB
- #define STARPU_CLUSTER_MAX_NB
- #define STARPU_CLUSTER_NB
- #define STARPU_CLUSTER_PREFERE_MIN
- #define STARPU_CLUSTER_KEEP_HOMOGENEOUS
- #define STARPU_CLUSTER_POLICY_NAME
- #define STARPU_CLUSTER_POLICY_STRUCT
- #define STARPU_CLUSTER_CREATE_FUNC
- #define STARPU_CLUSTER_CREATE_FUNC_ARG
- #define STARPU_CLUSTER_TYPE
- #define STARPU_CLUSTER_AWAKE_WORKERS
- #define STARPU_CLUSTER_PARTITION_ONE
- #define STARPU_CLUSTER_NEW
- #define STARPU_CLUSTER_NCORES

Enumerations

- enum starpu_parallel_worker_types { STARPU_PARALLEL_WORKER_OPENMP , STARPU_PARALLEL_WORKER_INTEL_O , STARPU_PARALLEL_WORKER_GNU_OPENMP_MKL }
- enum starpu_cluster_types { STARPU_CLUSTER_OPENMP , STARPU_CLUSTER_INTEL_OPENMP_MKL , STARPU_CLUSTER_GNU_OPENMP_MKL }

Functions

- struct starpu_parallel_worker_config ∗ starpu_parallel_worker_init (hwloc_obj_type_t parallel_worker_level,...)
- int starpu_parallel_worker_shutdown (struct starpu_parallel_worker_config ∗parallel_workers)
- int starpu_parallel_worker_print (struct starpu_parallel_worker_config ∗parallel_workers)
- void starpu_parallel_worker_openmp_prologue (void ∗)
- void **starpu_parallel_worker_gnu_openmp_mkl_prologue** (void ∗)
- struct starpu_cluster_machine ∗ starpu_cluster_machine (hwloc_obj_type_t cluster_level,...)
- int starpu_uncluster_machine (struct starpu_cluster_machine ∗clusters)
- int starpu_cluster_print (struct starpu_cluster_machine ∗clusters)

57.27.1 Detailed Description

57.27.2 Macro Definition Documentation

57.27.2.1 STARPU_PARALLEL_WORKER_MIN_NB

```
#define STARPU_PARALLEL_WORKER_MIN_NB
```

Used when calling starpu_parallel_worker_init()

57.27.2.2 STARPU_PARALLEL_WORKER_MAX_NB

```
#define STARPU_PARALLEL_WORKER_MAX_NB
```

Used when calling starpu_parallel_worker_init()

57.27.2.3 STARPU_PARALLEL_WORKER_NB

```
#define STARPU_PARALLEL_WORKER_NB
```

Used when calling starpu_parallel_worker_init()

57.27.2.4 STARPU_PARALLEL_WORKER_PREFERE_MIN

```
#define STARPU_PARALLEL_WORKER_PREFERE_MIN
```

Used when calling starpu_parallel_worker_init()

57.27.2.5 STARPU_PARALLEL_WORKER_KEEP_HOMOGENEOUS

```
#define STARPU_PARALLEL_WORKER_KEEP_HOMOGENEOUS
```

Used when calling starpu_parallel_worker_init()

57.27.2.6 STARPU_PARALLEL_WORKER_POLICY_NAME

```
#define STARPU_PARALLEL_WORKER_POLICY_NAME
```

Used when calling starpu_parallel_worker_init()

57.27.2.7 STARPU_PARALLEL_WORKER_POLICY_STRUCT

```
#define STARPU_PARALLEL_WORKER_POLICY_STRUCT
```

Used when calling starpu_parallel_worker_init()

57.27.2.8 STARPU_PARALLEL_WORKER_CREATE_FUNC

```
#define STARPU_PARALLEL_WORKER_CREATE_FUNC
```

Used when calling starpu_parallel_worker_init()

57.27.2.9 STARPU_PARALLEL_WORKER_CREATE_FUNC_ARG

```
#define STARPU_PARALLEL_WORKER_CREATE_FUNC_ARG
```

Used when calling starpu_parallel_worker_init()

57.27.2.10 STARPU_PARALLEL_WORKER_TYPE

```
#define STARPU_PARALLEL_WORKER_TYPE
```

Used when calling starpu_parallel_worker_init()

57.27.2.11 STARPU_PARALLEL_WORKER_AWAKE_WORKERS

```
#define STARPU_PARALLEL_WORKER_AWAKE_WORKERS
```

Used when calling starpu_parallel_worker_init()

57.27.2.12 STARPU_PARALLEL_WORKER_PARTITION_ONE

```
#define STARPU_PARALLEL_WORKER_PARTITION_ONE
```

Used when calling starpu_parallel_worker_init()

57.27.2.13 STARPU_PARALLEL_WORKER_NEW

```
#define STARPU_PARALLEL_WORKER_NEW
```

Used when calling starpu_parallel_worker_init()

57.27.2.14 STARPU_PARALLEL_WORKER_NCORES

```
#define STARPU_PARALLEL_WORKER_NCORES
```

Used when calling starpu_parallel_worker_init()

57.27.2.15 STARPU_CLUSTER_MIN_NB

```
#define STARPU_CLUSTER_MIN_NB
```

Deprecated Use STARPU_PARALLEL_WORKER_MIN_NB

57.27.2.16 STARPU_CLUSTER_MAX_NB

```
#define STARPU_CLUSTER_MAX_NB
```

Deprecated Use STARPU_PARALLEL_WORKER_MAX_NB

57.27.2.17 STARPU_CLUSTER_NB

```
#define STARPU_CLUSTER_NB
```

Deprecated Use STARPU_PARALLEL_WORKER_NB

57.27.2.18 STARPU_CLUSTER_PREFERE_MIN

```
#define STARPU_CLUSTER_PREFERE_MIN
```

Deprecated Use STARPU_PARALLEL_WORKER_PREFERE_MIN

57.27.2.19 STARPU_CLUSTER_KEEP_HOMOGENEOUS

```
#define STARPU_CLUSTER_KEEP_HOMOGENEOUS
```

Deprecated Use STARPU_PARALLEL_WORKER_KEEP_HOMOGENEOUS

57.27.2.20 STARPU_CLUSTER_POLICY_NAME

```
#define STARPU_CLUSTER_POLICY_NAME
```

Deprecated Use STARPU_PARALLEL_WORKER_POLICY_NAME

57.27.2.21 STARPU_CLUSTER_POLICY_STRUCT

```
#define STARPU_CLUSTER_POLICY_STRUCT
```

Deprecated Use STARPU_PARALLEL_WORKER_POLICY_STRUCT

57.27.2.22 STARPU_CLUSTER_CREATE_FUNC

`#define STARPU_CLUSTER_CREATE_FUNC`

Deprecated Use STARPU_PARALLEL_WORKER_CREATE_FUNC

57.27.2.23 STARPU_CLUSTER_CREATE_FUNC_ARG

`#define STARPU_CLUSTER_CREATE_FUNC_ARG`

Deprecated Use STARPU_PARALLEL_WORKER_CREATE_FUNC_ARG

57.27.2.24 STARPU_CLUSTER_TYPE

`#define STARPU_CLUSTER_TYPE`

Deprecated Use STARPU_PARALLEL_WORKER_TYPE

57.27.2.25 STARPU_CLUSTER_AWAKE_WORKERS

`#define STARPU_CLUSTER_AWAKE_WORKERS`

Deprecated Use STARPU_PARALLEL_WORKER_AWAKE_WORKERS

57.27.2.26 STARPU_CLUSTER_PARTITION_ONE

`#define STARPU_CLUSTER_PARTITION_ONE`

Deprecated Use STARPU_PARALLEL_WORKER_PARTITION_ONE

57.27.2.27 STARPU_CLUSTER_NEW

`#define STARPU_CLUSTER_NEW`

Deprecated Use STARPU_PARALLEL_WORKER_NEW

57.27.2.28 STARPU_CLUSTER_NCORES

`#define STARPU_CLUSTER_NCORES`

Deprecated Use STARPU_PARALLEL_WORKER_NCORES

57.27.3 Enumeration Type Documentation

57.27.3.1 starpu_parallel_worker_types

`enum starpu_parallel_worker_types`

These represent the default available functions to enforce parallel_worker use by the sub-runtime

Enumerator

STARPU_PARALLEL_WORKER_OPENMP	todo
STARPU_PARALLEL_WORKER_INTEL_OPENMP_MKL	todo
STARPU_PARALLEL_WORKER_GNU_OPENMP_MKL	todo

57.27.3.2 starpu_cluster_types

```
enum starpu_cluster_types
```

Deprecated Use starpu_parallel_worker_types

Enumerator

STARPU_CLUSTER_OPENMP	deprecated
STARPU_CLUSTER_INTEL_OPENMP_MKL	deprecated
STARPU_CLUSTER_GNU_OPENMP_MKL	deprecated

57.27.4 Function Documentation

57.27.4.1 starpu_parallel_worker_init()

```
struct starpu_parallel_worker_config * starpu_parallel_worker_init (
             hwloc_obj_type_t parallel_worker_level,
              ... )
```

Create parallel_workers on the machine with the given parameters. See Workers Creating Parallel Workers for more details.

57.27.4.2 starpu_parallel_worker_shutdown()

```
int starpu_parallel_worker_shutdown (
             struct starpu_parallel_worker_config * parallel_workers )
```

Delete the given parallel_workers configuration

57.27.4.3 starpu_parallel_worker_print()

```
int starpu_parallel_worker_print (
             struct starpu_parallel_worker_config * parallel_workers )
```

Print the given parallel_workers configuration. See Workers Creating Parallel Workers for more details.

57.27.4.4 starpu_parallel_worker_openmp_prologue()

```
void starpu_parallel_worker_openmp_prologue (
             void * )
```

Prologue functions

57.27.4.5 starpu_cluster_machine()

```
struct starpu_cluster_machine * starpu_cluster_machine (
             hwloc_obj_type_t cluster_level,
              ... )
```

Deprecated Use starpu_parallel_worker_init()

57.27.4.6 starpu_uncluster_machine()

```
int starpu_uncluster_machine (
            struct starpu_cluster_machine * clusters )
```

Deprecated Use starpu_parallel_worker_shutdown()

57.27.4.7 starpu_cluster_print()

```
int starpu_cluster_print (
            struct starpu_cluster_machine * clusters )
```

Deprecated Use starpu_parallel_worker_print()

57.28 Performance Monitoring Counters

API to access performance monitoring counters.

API

- enum starpu_perf_counter_scope { starpu_perf_counter_scope_undefined , starpu_perf_counter_scope_global , starpu_perf_counter_scope_per_worker , starpu_perf_counter_scope_per_codelet }
- enum starpu_perf_counter_type {
  starpu_perf_counter_type_undefined , starpu_perf_counter_type_int32 , starpu_perf_counter_type_int64 ,
  starpu_perf_counter_type_float ,
  starpu_perf_counter_type_double }
- void starpu_perf_counter_collection_start (void)
- void starpu_perf_counter_collection_stop (void)

Scope Related Routines

- int starpu_perf_counter_scope_name_to_id (const char ∗name)
- const char ∗ starpu_perf_counter_scope_id_to_name (enum starpu_perf_counter_scope scope)

Type Related Routines

- int starpu_perf_counter_type_name_to_id (const char ∗name)
- const char ∗ starpu_perf_counter_type_id_to_name (enum starpu_perf_counter_type type)

Counter Related Routines

- int starpu_perf_counter_nb (enum starpu_perf_counter_scope scope)
- int starpu_perf_counter_name_to_id (enum starpu_perf_counter_scope scope, const char ∗name)
- int starpu_perf_counter_nth_to_id (enum starpu_perf_counter_scope scope, int nth)
- const char ∗ starpu_perf_counter_id_to_name (int id)
- int starpu_perf_counter_get_type_id (int id)
- const char ∗ starpu_perf_counter_get_help_string (int id)

Listener Related Routines

- void starpu_perf_counter_list_avail (enum starpu_perf_counter_scope scope)
- void starpu_perf_counter_list_all_avail (void)
- struct starpu_perf_counter_set ∗ starpu_perf_counter_set_alloc (enum starpu_perf_counter_scope scope)
- void starpu_perf_counter_set_free (struct starpu_perf_counter_set ∗set)
- void starpu_perf_counter_set_enable_id (struct starpu_perf_counter_set ∗set, int id)
- void starpu_perf_counter_set_disable_id (struct starpu_perf_counter_set ∗set, int id)
- struct starpu_perf_counter_listener ∗ starpu_perf_counter_listener_init (struct starpu_perf_counter_set ∗set, void(∗callback)(struct starpu_perf_counter_listener ∗listener, struct starpu_perf_counter_sample ∗sample, void ∗context), void ∗user_arg)
- void starpu_perf_counter_listener_exit (struct starpu_perf_counter_listener ∗listener)
- void starpu_perf_counter_set_global_listener (struct starpu_perf_counter_listener ∗listener)
- void starpu_perf_counter_set_per_worker_listener (unsigned workerid, struct starpu_perf_counter_listener ∗listener)
- void starpu_perf_counter_set_all_per_worker_listeners (struct starpu_perf_counter_listener ∗listener)
- void starpu_perf_counter_set_per_codelet_listener (struct starpu_codelet ∗cl, struct starpu_perf_counter_listener ∗listener)
- void starpu_perf_counter_unset_global_listener (void)
- void starpu_perf_counter_unset_per_worker_listener (unsigned workerid)
- void starpu_perf_counter_unset_all_per_worker_listeners (void)
- void starpu_perf_counter_unset_per_codelet_listener (struct starpu_codelet ∗cl)

Sample Related Routines

- int32_t starpu_perf_counter_sample_get_int32_value (struct starpu_perf_counter_sample ∗sample, const int counter_id)
- int64_t starpu_perf_counter_sample_get_int64_value (struct starpu_perf_counter_sample ∗sample, const int counter_id)
- float starpu_perf_counter_sample_get_float_value (struct starpu_perf_counter_sample ∗sample, const int counter_id)
- double starpu_perf_counter_sample_get_double_value (struct starpu_perf_counter_sample ∗sample, const int counter_id)

57.28.1 Detailed Description

API to access performance monitoring counters.

57.28.2 Enumeration Type Documentation

57.28.2.1 starpu_perf_counter_scope

`enum starpu_perf_counter_scope`

Enum of all possible performance counter scopes.

Enumerator

starpu_perf_counter_scope_undefined	undefined scope
starpu_perf_counter_scope_global	global scope
starpu_perf_counter_scope_per_worker	per-worker scope
starpu_perf_counter_scope_per_codelet	per-codelet scope

57.28.2.2 starpu_perf_counter_type

`enum starpu_perf_counter_type`

Enum of all possible performance counter value type.

Enumerator

starpu_perf_counter_type_undefined	undefined value type
starpu_perf_counter_type_int32	signed 32-bit integer value
starpu_perf_counter_type_int64	signed 64-bit integer value
starpu_perf_counter_type_float	32-bit single precision floating-point value
starpu_perf_counter_type_double	64-bit double precision floating-point value

57.28.3 Function Documentation

57.28.3.1 starpu_perf_counter_collection_start()

```
void starpu_perf_counter_collection_start (
            void )
```

Start collecting performance counter values.

57.28.3.2 starpu_perf_counter_collection_stop()

```
void starpu_perf_counter_collection_stop (
             void   )
```

Stop collecting performance counter values.

57.28.3.3 starpu_perf_counter_scope_name_to_id()

```
int starpu_perf_counter_scope_name_to_id (
             const char * name )
```

Translate scope name constant string to scope id.

57.28.3.4 starpu_perf_counter_scope_id_to_name()

```
const char * starpu_perf_counter_scope_id_to_name (
             enum starpu_perf_counter_scope scope )
```

Translate scope id to scope name constant string.

57.28.3.5 starpu_perf_counter_type_name_to_id()

```
int starpu_perf_counter_type_name_to_id (
             const char * name )
```

Translate type name constant string to type id.

57.28.3.6 starpu_perf_counter_type_id_to_name()

```
const char * starpu_perf_counter_type_id_to_name (
             enum starpu_perf_counter_type type )
```

Translate type id to type name constant string.

57.28.3.7 starpu_perf_counter_nb()

```
int starpu_perf_counter_nb (
             enum starpu_perf_counter_scope scope )
```

Return the number of performance counters for the given scope.

57.28.3.8 starpu_perf_counter_name_to_id()

```
int starpu_perf_counter_name_to_id (
             enum starpu_perf_counter_scope scope,
             const char * name )
```

Translate a performance counter name to its id.

57.28.3.9 starpu_perf_counter_nth_to_id()

```
int starpu_perf_counter_nth_to_id (
             enum starpu_perf_counter_scope scope,
             int nth )
```

Translate a performance counter rank in its scope to its counter id.

57.28.3.10 starpu_perf_counter_id_to_name()

```
const char * starpu_perf_counter_id_to_name (
             int id )
```

Translate a counter id to its name constant string.

57.28.3.11 starpu_perf_counter_get_type_id()

```
int starpu_perf_counter_get_type_id (
             int id )
```

Return the counter's type id.

57.28.3.12 starpu_perf_counter_get_help_string()

```
const char * starpu_perf_counter_get_help_string (
            int id )
```

Return the counter's help string.

57.28.3.13 starpu_perf_counter_list_avail()

```
void starpu_perf_counter_list_avail (
            enum starpu_perf_counter_scope scope )
```

Display the list of counters defined in the given scope.

57.28.3.14 starpu_perf_counter_list_all_avail()

```
void starpu_perf_counter_list_all_avail (
            void )
```

Display the list of counters defined in all scopes.

57.28.3.15 starpu_perf_counter_set_alloc()

```
struct starpu_perf_counter_set * starpu_perf_counter_set_alloc (
            enum starpu_perf_counter_scope scope )
```

Allocate a new performance counter set.

57.28.3.16 starpu_perf_counter_set_free()

```
void starpu_perf_counter_set_free (
            struct starpu_perf_counter_set * set )
```

Free a performance counter set.

57.28.3.17 starpu_perf_counter_set_enable_id()

```
void starpu_perf_counter_set_enable_id (
            struct starpu_perf_counter_set * set,
            int id )
```

Enable a given counter in the set.

57.28.3.18 starpu_perf_counter_set_disable_id()

```
void starpu_perf_counter_set_disable_id (
            struct starpu_perf_counter_set * set,
            int id )
```

Disable a given counter in the set.

57.28.3.19 starpu_perf_counter_listener_init()

```
struct starpu_perf_counter_listener * starpu_perf_counter_listener_init (
            struct starpu_perf_counter_set * set,
            void(*)(struct starpu_perf_counter_listener *listener, struct starpu_perf_counter↵
_sample *sample, void *context) callback,
            void * user_arg )
```

Initialize a new performance counter listener.

57.28.3.20 starpu_perf_counter_listener_exit()

```
void starpu_perf_counter_listener_exit (
            struct starpu_perf_counter_listener * listener )
```

End a performance counter listener.

57.28.3.21 starpu_perf_counter_set_global_listener()

```
void starpu_perf_counter_set_global_listener (
             struct starpu_perf_counter_listener * listener )
```

Set a listener for the global scope.

57.28.3.22 starpu_perf_counter_set_per_worker_listener()

```
void starpu_perf_counter_set_per_worker_listener (
             unsigned workerid,
             struct starpu_perf_counter_listener * listener )
```

Set a listener for the per_worker scope on a given worker.

57.28.3.23 starpu_perf_counter_set_all_per_worker_listeners()

```
void starpu_perf_counter_set_all_per_worker_listeners (
             struct starpu_perf_counter_listener * listener )
```

Set a common listener for all workers.

57.28.3.24 starpu_perf_counter_set_per_codelet_listener()

```
void starpu_perf_counter_set_per_codelet_listener (
             struct starpu_codelet * cl,
             struct starpu_perf_counter_listener * listener )
```

Set a per_codelet listener for a codelet.

57.28.3.25 starpu_perf_counter_unset_global_listener()

```
void starpu_perf_counter_unset_global_listener (
             void )
```

Unset the global listener.

57.28.3.26 starpu_perf_counter_unset_per_worker_listener()

```
void starpu_perf_counter_unset_per_worker_listener (
             unsigned workerid )
```

Unset the per_worker listener.

57.28.3.27 starpu_perf_counter_unset_all_per_worker_listeners()

```
void starpu_perf_counter_unset_all_per_worker_listeners (
             void )
```

Unset all per_worker listeners.

57.28.3.28 starpu_perf_counter_unset_per_codelet_listener()

```
void starpu_perf_counter_unset_per_codelet_listener (
             struct starpu_codelet * cl )
```

Unset a per_codelet listener.

57.28.3.29 starpu_perf_counter_sample_get_int32_value()

```
int32_t starpu_perf_counter_sample_get_int32_value (
             struct starpu_perf_counter_sample * sample,
             const int counter_id )
```

Read an int32 counter value from a sample.

57.28.3.30 starpu_perf_counter_sample_get_int64_value()

```
int64_t starpu_perf_counter_sample_get_int64_value (
            struct starpu_perf_counter_sample * sample,
            const int counter_id )
```

Read an int64 counter value from a sample.

57.28.3.31 starpu_perf_counter_sample_get_float_value()

```
float starpu_perf_counter_sample_get_float_value (
            struct starpu_perf_counter_sample * sample,
            const int counter_id )
```

Read a float counter value from a sample.

57.28.3.32 starpu_perf_counter_sample_get_double_value()

```
double starpu_perf_counter_sample_get_double_value (
            struct starpu_perf_counter_sample * sample,
            const int counter_id )
```

Read a double counter value from a sample.

57.29 Performance Model

Data Structures

- struct starpu_perfmodel_device
- struct starpu_perfmodel_arch
- struct starpu_perfmodel_history_entry
- struct starpu_perfmodel_history_list
- struct starpu_perfmodel_regression_model
- struct starpu_perfmodel_per_arch
- struct starpu_perfmodel

Macros

- #define **starpu_per_arch_perfmodel**

Typedefs

- typedef double(∗ **starpu_perfmodel_per_arch_cost_function**) (struct starpu_task ∗task, struct starpu_perfmodel_arch ∗arch, unsigned nimpl)
- typedef size_t(∗ **starpu_perfmodel_per_arch_size_base**) (struct starpu_task ∗task, struct starpu_perfmodel_arch ∗arch, unsigned nimpl)
- typedef struct _starpu_perfmodel_state ∗ **starpu_perfmodel_state_t**

Enumerations

- enum starpu_perfmodel_type {
  STARPU_PERFMODEL_INVALID , STARPU_PER_WORKER , STARPU_PER_ARCH , STARPU_COMMON
  ,
  STARPU_HISTORY_BASED , STARPU_REGRESSION_BASED , STARPU_NL_REGRESSION_BASED ,
  STARPU_MULTIPLE_REGRESSION_BASED }

Functions

- void starpu_perfmodel_init (struct starpu_perfmodel ∗model)
- int starpu_perfmodel_deinit (struct starpu_perfmodel ∗model)
- int starpu_energy_start (int workerid, enum starpu_worker_archtype archi)
- int starpu_energy_stop (struct starpu_perfmodel ∗model, struct starpu_task ∗task, unsigned nimpl, unsigned ntasks, int workerid, enum starpu_worker_archtype archi)
- int starpu_perfmodel_load_file (const char ∗filename, struct starpu_perfmodel ∗model)
- int starpu_perfmodel_load_symbol (const char ∗symbol, struct starpu_perfmodel ∗model)
- int starpu_perfmodel_unload_model (struct starpu_perfmodel ∗model)
- void starpu_save_history_based_model (struct starpu_perfmodel ∗model)
- void starpu_perfmodel_get_model_path (const char ∗symbol, char ∗path, size_t maxlen)
- void starpu_perfmodel_dump_xml (FILE ∗output, struct starpu_perfmodel ∗model)
- void starpu_perfmodel_free_sampling (void)
- struct starpu_perfmodel_arch ∗ starpu_worker_get_perf_archtype (int workerid, unsigned sched_ctx_id)
- int **starpu_perfmodel_get_narch_combs** (void)
- int **starpu_perfmodel_arch_comb_add** (int ndevices, struct starpu_perfmodel_device ∗devices)
- int **starpu_perfmodel_arch_comb_get** (int ndevices, struct starpu_perfmodel_device ∗devices)
- struct starpu_perfmodel_arch ∗ **starpu_perfmodel_arch_comb_fetch** (int comb)
- struct starpu_perfmodel_per_arch ∗ **starpu_perfmodel_get_model_per_arch** (struct starpu_perfmodel ∗model, struct starpu_perfmodel_arch ∗arch, unsigned impl)
- struct starpu_perfmodel_per_arch ∗ **starpu_perfmodel_get_model_per_devices** (struct starpu_perfmodel ∗model, int impl,...)
- int **starpu_perfmodel_set_per_devices_cost_function** (struct starpu_perfmodel ∗model, int impl, starpu_perfmodel_per_arch_cost_function func,...)

- int **starpu_perfmodel_set_per_devices_size_base** (struct starpu_perfmodel ∗model, int impl, starpu_perfmodel_per_arch_size_base func,...)
- void starpu_perfmodel_debugfilepath (struct starpu_perfmodel ∗model, struct starpu_perfmodel_arch ∗arch, char ∗path, size_t maxlen, unsigned nimpl)
- const char ∗ **starpu_perfmodel_get_archtype_name** (enum starpu_worker_archtype archtype)
- void starpu_perfmodel_get_arch_name (struct starpu_perfmodel_arch ∗arch, char ∗archname, size_t maxlen, unsigned nimpl)
- double starpu_perfmodel_history_based_expected_perf (struct starpu_perfmodel ∗model, struct starpu_perfmodel_arch ∗arch, uint32_t footprint)
- void starpu_perfmodel_initialize (void)
- int starpu_perfmodel_list (FILE ∗output)
- void **starpu_perfmodel_print** (struct starpu_perfmodel ∗model, struct starpu_perfmodel_arch ∗arch, unsigned nimpl, char ∗parameter, uint32_t ∗footprint, FILE ∗output)
- int **starpu_perfmodel_print_all** (struct starpu_perfmodel ∗model, char ∗arch, char ∗parameter, uint32_t ∗footprint, FILE ∗output)
- int **starpu_perfmodel_print_estimations** (struct starpu_perfmodel ∗model, uint32_t footprint, FILE ∗output)
- int **starpu_perfmodel_list_combs** (FILE ∗output, struct starpu_perfmodel ∗model)
- void starpu_perfmodel_update_history (struct starpu_perfmodel ∗model, struct starpu_task ∗task, struct starpu_perfmodel_arch ∗arch, unsigned cpuid, unsigned nimpl, double measured)
- void starpu_perfmodel_update_history_n (struct starpu_perfmodel ∗model, struct starpu_task ∗task, struct starpu_perfmodel_arch ∗arch, unsigned cpuid, unsigned nimpl, double average_measured, unsigned number)
- void starpu_perfmodel_directory (FILE ∗output)
- void starpu_bus_print_bandwidth (FILE ∗f)
- void starpu_bus_print_affinity (FILE ∗f)
- void starpu_bus_print_filenames (FILE ∗f)
- double starpu_transfer_bandwidth (unsigned src_node, unsigned dst_node)
- double starpu_transfer_latency (unsigned src_node, unsigned dst_node)
- double starpu_transfer_predict (unsigned src_node, unsigned dst_node, size_t size)

Variables

- struct starpu_perfmodel starpu_perfmodel_nop

57.29.1 Detailed Description

57.29.2 Data Structure Documentation

57.29.2.1 struct starpu_perfmodel_device

todo

Data Fields

enum starpu_worker_archtype	type	type of the device
int	devid	identifier of the precise device
int	ncores	number of execution in parallel, minus 1

57.29.2.2 struct starpu_perfmodel_arch

todo

Data Fields

int	ndevices	number of the devices for the given arch
struct starpu_perfmodel_device ∗	devices	list of the devices for the given arch

57.29.2.3 struct starpu_perfmodel_history_entry

todo

Data Fields

double	mean	mean_n = 1/n sum
double	deviation	n dev_n = sum2 - 1/n (sum)^2
double	sum	sum of samples (in µs)
double	sum2	sum of samples^2
unsigned	nsample	number of samples
unsigned	nerror	
uint32_t	footprint	data footprint
size_t	size	in bytes
double	flops	Provided by the application
double	duration	
starpu_tag_t	tag	
double ∗	parameters	

57.29.2.4 struct starpu_perfmodel_history_list

todo

Data Fields

struct starpu_perfmodel_history_list ∗	next	
struct starpu_perfmodel_history_entry ∗	entry	

57.29.2.5 struct starpu_perfmodel_regression_model

todo

Data Fields

double	sumlny	sum of ln(measured)
double	sumlnx	sum of ln(size)
double	sumlnx2	sum of ln(size)^2
unsigned long	minx	minimum size
unsigned long	maxx	maximum size
double	sumlnxlny	sum of ln(size)∗ln(measured)
double	alpha	estimated = alpha ∗ size ^ beta
double	beta	estimated = alpha ∗ size ^ beta
unsigned	valid	whether the linear regression model is valid (i.e. enough measures)
double	a	estimated = a size ^b + c
double	b	estimated = a size ^b + c
double	c	estimated = a size ^b + c
unsigned	nl_valid	whether the non-linear regression model is valid (i.e. enough measures)
unsigned	nsample	number of sample values for non-linear regression
double ∗	coeff	list of computed coefficients for multiple linear regression model
unsigned	ncoeff	number of coefficients for multiple linear regression model
unsigned	multi_valid	whether the multiple linear regression model is valid

57.29.2.6 struct starpu_perfmodel_per_arch

information about the performance model of a given arch.

Data Fields

- starpu_perfmodel_per_arch_cost_function cost_function
- starpu_perfmodel_per_arch_size_base size_base
- char **debug_path** [256]

Private Attributes

- struct starpu_perfmodel_history_table ∗ history
- struct starpu_perfmodel_history_list ∗ list
- struct starpu_perfmodel_regression_model regression

57.29.2.6.1 Field Documentation

57.29.2.6.1.1 cost_function `starpu_perfmodel_per_arch_cost_function starpu_perfmodel_per_arch::cost_function`

Used by STARPU_PER_ARCH, must point to functions which take a task, the target arch and implementation number (as mere conveniency, since the array is already indexed by these), and must return a task duration estimation in micro-seconds.

57.29.2.6.1.2 size_base `starpu_perfmodel_per_arch_size_base starpu_perfmodel_per_arch::size_base`

Same as in structure starpu_perfmodel, but per-arch, in case it depends on the architecture-specific implementation.

57.29.2.6.1.3 history `struct starpu_perfmodel_history_table* starpu_perfmodel_per_arch::history [private]`

The history of performance measurements.

57.29.2.6.1.4 list `struct starpu_perfmodel_history_list* starpu_perfmodel_per_arch::list [private]`

Used by STARPU_HISTORY_BASED, STARPU_NL_REGRESSION_BASED and STARPU_MULTIPLE_REGRESSION_BASED, records all execution history measures.

57.29.2.6.1.5 regression `struct starpu_perfmodel_regression_model starpu_perfmodel_per_arch::regression [private]`

Used by STARPU_REGRESSION_BASED, STARPU_NL_REGRESSION_BASED and STARPU_MULTIPLE_REGRESSION_BASEI contains the estimated factors of the regression.

57.29.2.7 struct starpu_perfmodel

Contain all information about a performance model. At least the type and symbol fields have to be filled when defining a performance model for a codelet. For compatibility, make sure to initialize the whole structure to zero, either by using explicit memset, or by letting the compiler implicitly do it in e.g. static storage case. If not provided, other fields have to be zero.

Data Fields

- enum starpu_perfmodel_type type
- double(∗ cost_function)(struct starpu_task ∗, unsigned nimpl)
- double(∗ arch_cost_function)(struct starpu_task ∗, struct starpu_perfmodel_arch ∗arch, unsigned nimpl)
- double(∗ worker_cost_function)(struct starpu_task ∗, unsigned workerid, unsigned nimpl)
- size_t(∗ size_base)(struct starpu_task ∗, unsigned nimpl)
- uint32_t(∗ footprint)(struct starpu_task ∗)

- const char ∗ symbol
- char ∗ path
- void(∗ **parameters**)(struct starpu_task ∗task, double ∗parameters)

Private Attributes

- unsigned is_loaded
- unsigned **benchmarking**
- unsigned **is_init**
- const char ∗∗ parameters_names
- unsigned nparameters
- unsigned ∗∗ combinations
- unsigned ncombinations
- starpu_perfmodel_state_t **state**

57.29.2.7.1 Field Documentation

57.29.2.7.1.1 type `enum starpu_perfmodel_type starpu_perfmodel::type`

type of performance model

- STARPU_HISTORY_BASED, STARPU_REGRESSION_BASED, STARPU_NL_REGRESSION_BASED: No other fields needs to be provided, this is purely history-based.
- STARPU_MULTIPLE_REGRESSION_BASED: Need to provide fields starpu_perfmodel::nparameters (number of different parameters), starpu_perfmodel::ncombinations (number of parameters combinations-tuples) and table starpu_perfmodel::combinations which defines exponents of the equation. Function cl_perf_func also needs to define how to extract parameters from the task.
- STARPU_PER_ARCH: either field starpu_perfmodel::arch_cost_function has to be filled with a function that returns the cost in micro-seconds on the arch given as parameter, or field starpu_perfmodel::per_arch has to be filled with functions which return the cost in micro-seconds.
- STARPU_COMMON: field starpu_perfmodel::cost_function has to be filled with a function that returns the cost in micro-seconds on a CPU, timing on other archs will be determined by multiplying by an arch-specific factor.

57.29.2.7.1.2 cost_function `double(∗ starpu_perfmodel::cost_function) (struct starpu_task ∗, unsigned nimpl)`

Used by STARPU_COMMON. Take a task and implementation number, and must return a task duration estimation in micro-seconds.

57.29.2.7.1.3 arch_cost_function `double(∗ starpu_perfmodel::arch_cost_function) (struct starpu_task ∗, struct starpu_perfmodel_arch ∗arch, unsigned nimpl)`

Used by STARPU_PER_ARCH. Take a task, an arch and implementation number, and must return a task duration estimation in micro-seconds on that arch.

57.29.2.7.1.4 worker_cost_function `double(∗ starpu_perfmodel::worker_cost_function) (struct starpu_task ∗, unsigned workerid, unsigned nimpl)`

Used by STARPU_PER_WORKER. Take a task, a worker id and implementation number, and must return a task duration estimation in micro-seconds on that worker.

57.29.2.7.1.5 size_base `size_t(∗ starpu_perfmodel::size_base) (struct starpu_task ∗, unsigned nimpl)`

Used by STARPU_HISTORY_BASED, STARPU_REGRESSION_BASED and STARPU_NL_REGRESSION_BASED. If not `NULL`, take a task and implementation number, and return the size to be used as index to distinguish histories and as a base for regressions.

57.29.2.7.1.6 footprint `uint32_t(* starpu_perfmodel::footprint) (struct starpu_task *)`

Used by STARPU_HISTORY_BASED. If not `NULL`, take a task and return the footprint to be used as index to distinguish histories. The default is to use the starpu_task_data_footprint() function.

57.29.2.7.1.7 symbol `const char* starpu_perfmodel::symbol`

symbol name for the performance model, which will be used as file name to store the model. It must be set otherwise the model will be ignored.

57.29.2.7.1.8 path `char* starpu_perfmodel::path`

name of the file storing the performance model. It is non NULL if the model has been loaded or stored in a file.

57.29.2.7.1.9 is_loaded `unsigned starpu_perfmodel::is_loaded [private]`

Whether the performance model is already loaded from the disk.

57.29.2.7.1.10 parameters_names `const char** starpu_perfmodel::parameters_names [private]`

Names of parameters used for multiple linear regression models (M, N, K)

57.29.2.7.1.11 nparameters `unsigned starpu_perfmodel::nparameters [private]`

Number of parameters used for multiple linear regression models

57.29.2.7.1.12 combinations `unsigned** starpu_perfmodel::combinations [private]`

Table of combinations of parameters (and the exponents) used for multiple linear regression models

57.29.2.7.1.13 ncombinations `unsigned starpu_perfmodel::ncombinations [private]`

Number of combination of parameters used for multiple linear regression models

57.29.3 Enumeration Type Documentation

57.29.3.1 starpu_perfmodel_type

`enum starpu_perfmodel_type`

todo

Enumerator

STARPU_PER_WORKER	Application-provided per-worker cost model function
STARPU_PER_ARCH	Application-provided per-arch cost model function
STARPU_COMMON	Application-provided common cost model function, with per-arch factor
STARPU_HISTORY_BASED	Automatic history-based cost model
STARPU_REGRESSION_BASED	Automatic linear regression-based cost model (alpha ∗ size ^ beta)
STARPU_NL_REGRESSION_BASED	Automatic non-linear regression-based cost model (a ∗ size ^ b + c)
STARPU_MULTIPLE_REGRESSION_BASED	Automatic multiple linear regression-based cost model. Application provides parameters, their combinations and exponents.

57.29.4 Function Documentation

57.29.4.1 starpu_perfmodel_init()

```
void starpu_perfmodel_init (
            struct starpu_perfmodel * model )
```

Initialize the `model` performance model structure. This is automatically called when e.g. submitting a task using a codelet using this performance model.

57.29.4.2 starpu_perfmodel_deinit()

```
int starpu_perfmodel_deinit (
            struct starpu_perfmodel * model )
```

Deinitialize the `model` performance model structure. You need to call this before deallocating the structure. You will probably want to call starpu_perfmodel_unload_model() before calling this function, to save the perfmodel.

57.29.4.3 starpu_energy_start()

```
int starpu_energy_start (
            int workerid,
            enum starpu_worker_archtype archi )
```

starpu_energy_start - start counting hardware events in an event set

- `workerid` is the worker on which calibration is to be performed (in the case of GPUs, use -1 for CPUs)
- `archi` is the type of architecture on which calibration will be run

See Measuring energy and power with StarPU for more details.

57.29.4.4 starpu_energy_stop()

```
int starpu_energy_stop (
            struct starpu_perfmodel * model,
            struct starpu_task * task,
            unsigned nimpl,
            unsigned ntasks,
            int workerid,
            enum starpu_worker_archtype archi )
```

starpu_energy_stop - stop counting hardware events in an event set

- `model` is the energy performance model to be filled with the result
- `task` is a task specimen, so the performance model folds the result according to the parameter sizes of the task.
- `nimpl` is the implementation number run during calibration
- `ntasks` is the number of tasks run during calibration
- `workerid` is the worker on which calibration was performed (in the case of GPUs, use -1 for CPUs)
- `archi` is the type of architecture on which calibration was run

See Measuring energy and power with StarPU for more details.

57.29.4.5 starpu_perfmodel_load_file()

```
int starpu_perfmodel_load_file (
            const char * filename,
            struct starpu_perfmodel * model )
```

Load the performance model found in the file named `filename`. `model` has to be completely zero, and will be filled with the information stored in the given file.

57.29.4.6 starpu_perfmodel_load_symbol()

```
int starpu_perfmodel_load_symbol (
            const char * symbol,
            struct starpu_perfmodel * model )
```

Load a given performance model. `model` has to be completely zero, and will be filled with the information stored in `$STARPU_HOME/.starpu`. The function is intended to be used by external tools that want to read the performance model files.

57.29.4.7 starpu_perfmodel_unload_model()

```
int starpu_perfmodel_unload_model (
            struct starpu_perfmodel * model )
```

Unload `model` which has been previously loaded through the function starpu_perfmodel_load_symbol()

57.29.4.8 starpu_save_history_based_model()

```
void starpu_save_history_based_model (
            struct starpu_perfmodel * model )
```

Save the performance model in its file.

57.29.4.9 starpu_perfmodel_get_model_path()

```
void starpu_perfmodel_get_model_path (
            const char * symbol,
            char * path,
            size_t maxlen )
```

Fills `path` (supposed to be `maxlen` long) with the full path to the performance model file for symbol `symbol`. This path can later on be used for instance with starpu_perfmodel_load_file() .

57.29.4.10 starpu_perfmodel_dump_xml()

```
void starpu_perfmodel_dump_xml (
            FILE * output,
            struct starpu_perfmodel * model )
```

Dump performance model `model` to output stream `output`, in XML format. See Performance Model Example for more details.

57.29.4.11 starpu_perfmodel_free_sampling()

```
void starpu_perfmodel_free_sampling (
            void )
```

Free internal memory used for sampling management. It should only be called by an application which is not calling starpu_shutdown() as this function already calls it. See for example `tools/starpu_perfmodel_display.c`.

57.29.4.12 starpu_worker_get_perf_archtype()

```
struct starpu_perfmodel_arch * starpu_worker_get_perf_archtype (
            int workerid,
            unsigned sched_ctx_id )
```

Return the architecture type of the worker `workerid`.

57.29.4.13 starpu_perfmodel_debugfilepath()

```
void starpu_perfmodel_debugfilepath (
            struct starpu_perfmodel * model,
            struct starpu_perfmodel_arch * arch,
            char * path,
            size_t maxlen,
            unsigned nimpl )
```

Return the path to the debugging information for the performance model.

57.29.4.14 starpu_perfmodel_get_arch_name()

```
void starpu_perfmodel_get_arch_name (
            struct starpu_perfmodel_arch * arch,
            char * archname,
            size_t maxlen,
            unsigned nimpl )
```

Return the architecture name for `arch`

57.29.4.15 starpu_perfmodel_history_based_expected_perf()

```
double starpu_perfmodel_history_based_expected_perf (
            struct starpu_perfmodel * model,
            struct starpu_perfmodel_arch * arch,
            uint32_t footprint )
```

Return the estimated time in µs of a task with the given model and the given footprint.

57.29.4.16 starpu_perfmodel_initialize()

```
void starpu_perfmodel_initialize (
            void )
```

If starpu_init() is not used, starpu_perfmodel_initialize() should be used called calling starpu_perfmodel_∗ functions.

57.29.4.17 starpu_perfmodel_list()

```
int starpu_perfmodel_list (
            FILE * output )
```

Print a list of all performance models on `output`

57.29.4.18 starpu_perfmodel_update_history()

```
void starpu_perfmodel_update_history (
            struct starpu_perfmodel * model,
            struct starpu_task * task,
            struct starpu_perfmodel_arch * arch,
            unsigned cpuid,
            unsigned nimpl,
            double measured )
```

Feed the performance model `model` with one explicit measurement (in µs or J), in addition to measurements done by StarPU itself. This can be useful when the application already has an existing set of measurements done in good conditions, that StarPU could benefit from instead of doing on-line measurements. An example of use can be seen in Performance Model Example.

Note that this records only one measurement, and StarPU would ignore the first measurement (since it is usually disturbed by library loading etc.). Make sure to call this function several times to record all your measurements.

You can also call starpu_perfmodel_update_history_n() to directly provide an average performed on several tasks.

See Performance Model Calibration for more details.

57.29.4.19 starpu_perfmodel_update_history_n()

```
void starpu_perfmodel_update_history_n (
            struct starpu_perfmodel * model,
            struct starpu_task * task,
            struct starpu_perfmodel_arch * arch,
            unsigned cpuid,
            unsigned nimpl,
            double average_measured,
            unsigned number )
```

Feed the performance model `model` with an explicit average measurement (in µs or J).
This is similar to starpu_perfmodel_update_history(), but records a batch of `number` measurements provided as the average of the measurements `average_measured`.

57.29.4.20 starpu_perfmodel_directory()

```
void starpu_perfmodel_directory (
            FILE * output )
```

Print the directory name storing performance models on `output`

57.29.4.21 starpu_bus_print_bandwidth()

```
void starpu_bus_print_bandwidth (
            FILE * f )
```

Print a matrix of bus bandwidths on `f`.

57.29.4.22 starpu_bus_print_affinity()

```
void starpu_bus_print_affinity (
            FILE * f )
```

Print the affinity devices on `f`.

57.29.4.23 starpu_bus_print_filenames()

```
void starpu_bus_print_filenames (
            FILE * f )
```

Print on `f` the name of the files containing the matrix of bus bandwidths, the affinity devices and the latency.

57.29.4.24 starpu_transfer_bandwidth()

```
double starpu_transfer_bandwidth (
            unsigned src_node,
            unsigned dst_node )
```

Return the bandwidth of data transfer between two memory nodes. See Helper functions for defining a scheduling policy (Basic or mod
for more details.

57.29.4.25 starpu_transfer_latency()

```
double starpu_transfer_latency (
            unsigned src_node,
            unsigned dst_node )
```

Return the latency of data transfer between two memory nodes. See Helper functions for defining a scheduling policy (Basic or modula
for more details.

57.29.4.26 starpu_transfer_predict()

```
double starpu_transfer_predict (
            unsigned src_node,
            unsigned dst_node,
            size_t size )
```

Return the estimated time to transfer a given size between two memory nodes. See Helper functions for defining a scheduling policy (B
for more details.

57.29.5 Variable Documentation

57.29.5.1 starpu_perfmodel_nop

`struct starpu_perfmodel starpu_perfmodel_nop [extern]`

Performance model which just always return 1µs.

57.30 Performance Steering Knobs

API to access performance steering counters.

API

- enum starpu_perf_knob_scope { starpu_perf_knob_scope_undefined , starpu_perf_knob_scope_global , starpu_perf_knob_scope_per_worker , starpu_perf_knob_scope_per_scheduler }
- enum starpu_perf_knob_type {
  starpu_perf_knob_type_undefined , starpu_perf_knob_type_int32 , starpu_perf_knob_type_int64 ,
  starpu_perf_knob_type_float ,
  starpu_perf_knob_type_double }

Scope Related Routines

- int starpu_perf_knob_scope_name_to_id (const char ∗name)
- const char ∗ starpu_perf_knob_scope_id_to_name (enum starpu_perf_knob_scope scope)

Type Related Routines

- int starpu_perf_knob_type_name_to_id (const char ∗name)
- const char ∗ starpu_perf_knob_type_id_to_name (enum starpu_perf_knob_type type)

Performance Steering Knob Related Routines

- int starpu_perf_knob_nb (enum starpu_perf_knob_scope scope)
- int starpu_perf_knob_name_to_id (enum starpu_perf_knob_scope scope, const char ∗name)
- int starpu_perf_knob_nth_to_id (enum starpu_perf_knob_scope scope, int nth)
- const char ∗ starpu_perf_knob_id_to_name (int id)
- int starpu_perf_knob_get_type_id (int id)
- const char ∗ starpu_perf_knob_get_help_string (int id)
- void starpu_perf_knob_list_avail (enum starpu_perf_knob_scope scope)
- void starpu_perf_knob_list_all_avail (void)
- int32_t starpu_perf_knob_get_global_int32_value (const int knob_id)
- int64_t starpu_perf_knob_get_global_int64_value (const int knob_id)
- float starpu_perf_knob_get_global_float_value (const int knob_id)
- double starpu_perf_knob_get_global_double_value (const int knob_id)
- void starpu_perf_knob_set_global_int32_value (const int knob_id, int32_t new_value)
- void starpu_perf_knob_set_global_int64_value (const int knob_id, int64_t new_value)
- void starpu_perf_knob_set_global_float_value (const int knob_id, float new_value)
- void starpu_perf_knob_set_global_double_value (const int knob_id, double new_value)
- int32_t starpu_perf_knob_get_per_worker_int32_value (const int knob_id, unsigned workerid)
- int64_t starpu_perf_knob_get_per_worker_int64_value (const int knob_id, unsigned workerid)
- float starpu_perf_knob_get_per_worker_float_value (const int knob_id, unsigned workerid)
- double starpu_perf_knob_get_per_worker_double_value (const int knob_id, unsigned workerid)
- void starpu_perf_knob_set_per_worker_int32_value (const int knob_id, unsigned workerid, int32_t new_value)
- void starpu_perf_knob_set_per_worker_int64_value (const int knob_id, unsigned workerid, int64_t new_value)
- void starpu_perf_knob_set_per_worker_float_value (const int knob_id, unsigned workerid, float new_value)
- void starpu_perf_knob_set_per_worker_double_value (const int knob_id, unsigned workerid, double new_value)
- int32_t starpu_perf_knob_get_per_scheduler_int32_value (const int knob_id, const char ∗sched_policy_name)
- int64_t starpu_perf_knob_get_per_scheduler_int64_value (const int knob_id, const char ∗sched_policy_name)

- float starpu_perf_knob_get_per_scheduler_float_value (const int knob_id, const char ∗sched_policy_name)
- double starpu_perf_knob_get_per_scheduler_double_value (const int knob_id, const char ∗sched_policy_↩ name)
- void starpu_perf_knob_set_per_scheduler_int32_value (const int knob_id, const char ∗sched_policy_name, int32_t new_value)
- void starpu_perf_knob_set_per_scheduler_int64_value (const int knob_id, const char ∗sched_policy_name, int64_t new_value)
- void starpu_perf_knob_set_per_scheduler_float_value (const int knob_id, const char ∗sched_policy_name, float new_value)
- void starpu_perf_knob_set_per_scheduler_double_value (const int knob_id, const char ∗sched_policy_↩ name, double new_value)

57.30.1 Detailed Description

API to access performance steering counters.

57.30.2 Enumeration Type Documentation

57.30.2.1 starpu_perf_knob_scope

`enum starpu_perf_knob_scope`

Enum of all possible performance knob scopes.

Enumerator

starpu_perf_knob_scope_undefined	undefined scope
starpu_perf_knob_scope_global	global scope
starpu_perf_knob_scope_per_worker	per-worker scope
starpu_perf_knob_scope_per_scheduler	per-scheduler scope

57.30.2.2 starpu_perf_knob_type

`enum starpu_perf_knob_type`

Enum of all possible performance knob value type.

Enumerator

starpu_perf_knob_type_undefined	undefined value type
starpu_perf_knob_type_int32	signed 32-bit integer value
starpu_perf_knob_type_int64	signed 64-bit integer value
starpu_perf_knob_type_float	32-bit single precision floating-point value
starpu_perf_knob_type_double	64-bit double precision floating-point value

57.30.3 Function Documentation

57.30.3.1 starpu_perf_knob_scope_name_to_id()

```
int starpu_perf_knob_scope_name_to_id (
            const char * name )
```

Translate scope name constant string to scope id.

57.30.3.2 starpu_perf_knob_scope_id_to_name()

```
const char * starpu_perf_knob_scope_id_to_name (
             enum starpu_perf_knob_scope scope )
```

Translate scope id to scope name constant string.

57.30.3.3 starpu_perf_knob_type_name_to_id()

```
int starpu_perf_knob_type_name_to_id (
             const char * name )
```

Translate type name constant string to type id.

57.30.3.4 starpu_perf_knob_type_id_to_name()

```
const char * starpu_perf_knob_type_id_to_name (
             enum starpu_perf_knob_type type )
```

Translate type id to type name constant string.

57.30.3.5 starpu_perf_knob_nb()

```
int starpu_perf_knob_nb (
             enum starpu_perf_knob_scope scope )
```

Return the number of performance steering knobs for the given scope.

57.30.3.6 starpu_perf_knob_name_to_id()

```
int starpu_perf_knob_name_to_id (
             enum starpu_perf_knob_scope scope,
             const char * name )
```

Translate a performance knob name to its id.

57.30.3.7 starpu_perf_knob_nth_to_id()

```
int starpu_perf_knob_nth_to_id (
             enum starpu_perf_knob_scope scope,
             int nth )
```

Translate a performance knob name to its id.

57.30.3.8 starpu_perf_knob_id_to_name()

```
const char * starpu_perf_knob_id_to_name (
             int id )
```

Translate a performance knob rank in its scope to its knob id.

57.30.3.9 starpu_perf_knob_get_type_id()

```
int starpu_perf_knob_get_type_id (
             int id )
```

Translate a knob id to its name constant string.

57.30.3.10 starpu_perf_knob_get_help_string()

```
const char * starpu_perf_knob_get_help_string (
             int id )
```

Return the knob's help string.

57.30.3.11 starpu_perf_knob_list_avail()

```
void starpu_perf_knob_list_avail (
            enum starpu_perf_knob_scope scope )
```

Display the list of knobs defined in the given scope.

57.30.3.12 starpu_perf_knob_list_all_avail()

```
void starpu_perf_knob_list_all_avail (
            void )
```

Display the list of knobs defined in all scopes.

57.30.3.13 starpu_perf_knob_get_global_int32_value()

```
int32_t starpu_perf_knob_get_global_int32_value (
            const int knob_id )
```

Get knob value for Global scope.

57.30.3.14 starpu_perf_knob_get_global_int64_value()

```
int64_t starpu_perf_knob_get_global_int64_value (
            const int knob_id )
```

Get knob value for Global scope.

57.30.3.15 starpu_perf_knob_get_global_float_value()

```
float starpu_perf_knob_get_global_float_value (
            const int knob_id )
```

Get knob value for Global scope.

57.30.3.16 starpu_perf_knob_get_global_double_value()

```
double starpu_perf_knob_get_global_double_value (
            const int knob_id )
```

Get knob value for Global scope.

57.30.3.17 starpu_perf_knob_set_global_int32_value()

```
void starpu_perf_knob_set_global_int32_value (
            const int knob_id,
            int32_t new_value )
```

Set int32 knob value for Global scope.

57.30.3.18 starpu_perf_knob_set_global_int64_value()

```
void starpu_perf_knob_set_global_int64_value (
            const int knob_id,
            int64_t new_value )
```

Set int64 knob value for Global scope.

57.30.3.19 starpu_perf_knob_set_global_float_value()

```
void starpu_perf_knob_set_global_float_value (
            const int knob_id,
            float new_value )
```

Set float knob value for Global scope.

57.30.3.20 starpu_perf_knob_set_global_double_value()

```
void starpu_perf_knob_set_global_double_value (
            const int knob_id,
            double new_value )
```

Set double knob value for Global scope.

57.30.3.21 starpu_perf_knob_get_per_worker_int32_value()

```
int32_t starpu_perf_knob_get_per_worker_int32_value (
            const int knob_id,
            unsigned workerid )
```

Get int32 value for Per_worker scope.

57.30.3.22 starpu_perf_knob_get_per_worker_int64_value()

```
int64_t starpu_perf_knob_get_per_worker_int64_value (
            const int knob_id,
            unsigned workerid )
```

Get int64 value for Per_worker scope.

57.30.3.23 starpu_perf_knob_get_per_worker_float_value()

```
float starpu_perf_knob_get_per_worker_float_value (
            const int knob_id,
            unsigned workerid )
```

Get float value for Per_worker scope.

57.30.3.24 starpu_perf_knob_get_per_worker_double_value()

```
double starpu_perf_knob_get_per_worker_double_value (
            const int knob_id,
            unsigned workerid )
```

Get double value for Per_worker scope.

57.30.3.25 starpu_perf_knob_set_per_worker_int32_value()

```
void starpu_perf_knob_set_per_worker_int32_value (
            const int knob_id,
            unsigned workerid,
            int32_t new_value )
```

Set int32 value for Per_worker scope.

57.30.3.26 starpu_perf_knob_set_per_worker_int64_value()

```
void starpu_perf_knob_set_per_worker_int64_value (
            const int knob_id,
            unsigned workerid,
            int64_t new_value )
```

Set int64 value for Per_worker scope.

57.30.3.27 starpu_perf_knob_set_per_worker_float_value()

```
void starpu_perf_knob_set_per_worker_float_value (
            const int knob_id,
            unsigned workerid,
            float new_value )
```

Set float value for Per_worker scope.

57.30.3.28 starpu_perf_knob_set_per_worker_double_value()

```
void starpu_perf_knob_set_per_worker_double_value (
            const int knob_id,
            unsigned workerid,
            double new_value )
```

Set double value for Per_worker scope.

57.30.3.29 starpu_perf_knob_get_per_scheduler_int32_value()

```
int32_t starpu_perf_knob_get_per_scheduler_int32_value (
            const int knob_id,
            const char * sched_policy_name )
```

Get int32 value for per_scheduler scope.

57.30.3.30 starpu_perf_knob_get_per_scheduler_int64_value()

```
int64_t starpu_perf_knob_get_per_scheduler_int64_value (
            const int knob_id,
            const char * sched_policy_name )
```

Get int64 value for per_scheduler scope.

57.30.3.31 starpu_perf_knob_get_per_scheduler_float_value()

```
float starpu_perf_knob_get_per_scheduler_float_value (
            const int knob_id,
            const char * sched_policy_name )
```

Get float value for per_scheduler scope.

57.30.3.32 starpu_perf_knob_get_per_scheduler_double_value()

```
double starpu_perf_knob_get_per_scheduler_double_value (
            const int knob_id,
            const char * sched_policy_name )
```

Get double value for per_scheduler scope.

57.30.3.33 starpu_perf_knob_set_per_scheduler_int32_value()

```
void starpu_perf_knob_set_per_scheduler_int32_value (
            const int knob_id,
            const char * sched_policy_name,
            int32_t new_value )
```

Set int32 value for per_scheduler scope.

57.30.3.34 starpu_perf_knob_set_per_scheduler_int64_value()

```
void starpu_perf_knob_set_per_scheduler_int64_value (
            const int knob_id,
            const char * sched_policy_name,
            int64_t new_value )
```

Set int64 value for per_scheduler scope.

57.30.3.35 starpu_perf_knob_set_per_scheduler_float_value()

```
void starpu_perf_knob_set_per_scheduler_float_value (
            const int knob_id,
            const char * sched_policy_name,
            float new_value )
```

Set float value for per_scheduler scope.

57.30.3.36 starpu_perf_knob_set_per_scheduler_double_value()

```
void starpu_perf_knob_set_per_scheduler_double_value (
            const int knob_id,
            const char * sched_policy_name,
            double new_value )
```

Set double value for per_scheduler scope.

57.31 Profiling

Data Structures

- struct starpu_profiling_task_info
- struct starpu_profiling_worker_info
- struct starpu_profiling_bus_info

Macros

- #define STARPU_PROFILING_DISABLE
- #define STARPU_PROFILING_ENABLE
- #define **STARPU_NS_PER_S**
- #define **starpu_timespec_cmp**(a, b, CMP)

Functions

- void starpu_profiling_init (void)
- void starpu_profiling_set_id (int new_id)
- int starpu_profiling_status_set (int status)
- int starpu_profiling_status_get (void)
- int starpu_profiling_worker_get_info (int workerid, struct starpu_profiling_worker_info ∗worker_info)
- int starpu_bus_get_count (void)
- int starpu_bus_get_id (int src, int dst)
- int starpu_bus_get_src (int busid)
- int starpu_bus_get_dst (int busid)
- void starpu_bus_set_direct (int busid, int direct)
- int starpu_bus_get_direct (int busid)
- void starpu_bus_set_ngpus (int busid, int ngpus)
- int starpu_bus_get_ngpus (int busid)
- int starpu_bus_get_profiling_info (int busid, struct starpu_profiling_bus_info ∗bus_info)
- static __starpu_inline void **starpu_timespec_clear** (struct timespec ∗tsp)
- static __starpu_inline void **starpu_timespec_add** (struct timespec ∗a, struct timespec ∗b, struct timespec ∗result)
- static __starpu_inline void **starpu_timespec_accumulate** (struct timespec ∗result, struct timespec ∗a)
- static __starpu_inline void **starpu_timespec_sub** (const struct timespec ∗a, const struct timespec ∗b, struct timespec ∗result)
- double starpu_timing_timespec_delay_us (struct timespec ∗start, struct timespec ∗end)
- double starpu_timing_timespec_to_us (struct timespec ∗ts)
- void starpu_profiling_bus_helper_display_summary (void)
- void starpu_profiling_worker_helper_display_summary (void)
- void starpu_data_display_memory_stats (void)

57.31.1 Detailed Description

57.31.2 Data Structure Documentation

57.31.2.1 struct starpu_profiling_task_info

Information about the execution of a task. It is accessible from the field starpu_task::profiling_info if profiling was enabled.

Data Fields

struct timespec	submit_time	Date of task submission (relative to the initialization of StarPU).
struct timespec	push_start_time	Time when the task was submitted to the scheduler.
struct timespec	push_end_time	Time when the scheduler finished with the task submission.

Data Fields

struct timespec	pop_start_time	Time when the scheduler started to be requested for a task, and eventually gave that task.
struct timespec	pop_end_time	Time when the scheduler finished providing the task for execution.
struct timespec	acquire_data_start_time	Time when the worker started fetching input data.
struct timespec	acquire_data_end_time	Time when the worker finished fetching input data.
struct timespec	start_time	Date of task execution beginning (relative to the initialization of StarPU).
struct timespec	end_time	Date of task execution termination (relative to the initialization of StarPU).
struct timespec	release_data_start_time	Time when the worker started releasing data.
struct timespec	release_data_end_time	Time when the worker finished releasing data.
struct timespec	callback_start_time	Time when the worker started the application callback for the task.
struct timespec	callback_end_time	Time when the worker finished the application callback for the task.
int	workerid	Identifier of the worker which has executed the task.
uint64_t	used_cycles	Number of cycles used by the task, only available in the MoviSim
uint64_t	stall_cycles	Number of cycles stalled within the task, only available in the MoviSim
double	energy_consumed	Energy consumed by the task, in Joules

57.31.2.2 struct starpu_profiling_worker_info

Profiling information associated to a worker. The timing is provided since the previous call to starpu_profiling_worker_get_info(). The executing_time, callback_time, waiting_time, sleeping_time, and scheduling_time are exclusive to each other, i.e. they can be added up, their sum is smaller than total_time. The difference between total_time and the sum is the uncategorized runtime overhead.

Data Fields

struct timespec	start_time	Starting date for the reported profiling measurements.
struct timespec	total_time	Duration of the profiling measurement interval.
struct timespec	executing_time	Time spent by the worker to execute tasks during the profiling measurement interval.
struct timespec	callback_time	Time spent by the worker to execute callbacks, while not executing a task, during the profiling measurement interval.
struct timespec	waiting_time	Time spent by the worker waiting for a data transfer to finish, while not executing a task or a callback, during the profiling measurement interval.
struct timespec	sleeping_time	Time spent idling by the worker because no task were available, and not executing a task or a callback or waiting for a data transfer to finish, during the profiling measurement interval.
struct timespec	scheduling_time	Time spent by the worker scheduling tasks, while not executing a task or a callback or waiting for a data transfer to finish, and there are tasks to be scheduled, during the profiling measurement interval.
struct timespec	all_executing_time	Time spent by the worker to execute tasks during the profiling measurement interval. Normally always equal to executing_time.
struct timespec	all_callback_time	Time spent by the worker to execute callbacks during the profiling measurement interval. Normally always greater than callback_time.

Data Fields

struct timespec	all_waiting_time	Time spent by the worker waiting for a data transfer to finish during the profiling measurement interval. Normally always greater than waiting_time.
struct timespec	all_sleeping_time	Time spent idling by the worker because no task were available during the profiling measurement interval. Normally always greater than sleeping_time.
struct timespec	all_scheduling_time	Time spent by the worker scheduling tasks during the profiling measurement interval. Normally always greater than scheduling_time.
int	executed_tasks	Number of tasks executed by the worker during the profiling measurement interval.
uint64_t	used_cycles	Number of cycles used by the worker, only available in the MoviSim
uint64_t	stall_cycles	Number of cycles stalled within the worker, only available in the MoviSim
double	energy_consumed	Energy consumed by the worker, in Joules
double	flops	

57.31.2.3 struct starpu_profiling_bus_info

todo

Data Fields

struct timespec	start_time	Time of bus profiling startup.
struct timespec	total_time	Total time of bus profiling.
int long long	transferred_bytes	Number of bytes transferred during profiling.
int	transfer_count	Number of transfers during profiling.

57.31.3 Macro Definition Documentation

57.31.3.1 STARPU_PROFILING_DISABLE

```
#define STARPU_PROFILING_DISABLE
```

Used when calling the function starpu_profiling_status_set() to disable profiling.

57.31.3.2 STARPU_PROFILING_ENABLE

```
#define STARPU_PROFILING_ENABLE
```

Used when calling the function starpu_profiling_status_set() to enable profiling.

57.31.4 Function Documentation

57.31.4.1 starpu_profiling_init()

```
void starpu_profiling_init (
            void )
```

Reset performance counters and enable profiling if the environment variable STARPU_PROFILING is set to a positive value. See Enabling On-line Performance Monitoring for more details.

57.31.4.2 starpu_profiling_set_id()

```
void starpu_profiling_set_id (
            int new_id )
```

Set the ID used for profiling trace filename. Has to be called before starpu_init(). See Tracing MPI applications for more details.

57.31.4.3 starpu_profiling_status_set()

```
int starpu_profiling_status_set (
            int status )
```

Set the profiling status. Profiling is activated by passing STARPU_PROFILING_ENABLE in `status`. Passing STARPU_PROFILING_DISABLE disables profiling. Calling this function resets all profiling measurements. When profiling is enabled, the field starpu_task::profiling_info points to a valid structure starpu_profiling_task_info containing information about the execution of the task. Negative return values indicate an error, otherwise the previous status is returned. See Enabling On-line Performance Monitoring for more details.

57.31.4.4 starpu_profiling_status_get()

```
int starpu_profiling_status_get (
            void )
```

Return the current profiling status or a negative value in case there was an error. See Enabling On-line Performance Monitoring for more details.

57.31.4.5 starpu_profiling_worker_get_info()

```
int starpu_profiling_worker_get_info (
            int workerid,
            struct starpu_profiling_worker_info * worker_info )
```

Get the profiling info associated to the worker identified by `workerid`, and reset the profiling measurements. If the argument `worker_info` is `NULL`, only reset the counters associated to worker `workerid`. Upon successful completion, this function returns 0. Otherwise, a negative value is returned. See Per-worker Feedback for more details.

57.31.4.6 starpu_bus_get_count()

```
int starpu_bus_get_count (
            void )
```

Return the number of buses in the machine. See Hardware Topology for more details.

57.31.4.7 starpu_bus_get_id()

```
int starpu_bus_get_id (
            int src,
            int dst )
```

Return the identifier of the bus between `src` and `dst`. See Hardware Topology for more details.

57.31.4.8 starpu_bus_get_src()

```
int starpu_bus_get_src (
            int busid )
```

Return the source point of bus `busid`. See Hardware Topology for more details.

57.31.4.9 starpu_bus_get_dst()

```
int starpu_bus_get_dst (
            int busid )
```

Return the destination point of bus `busid`. See Hardware Topology for more details.

57.31.4.10 starpu_bus_set_direct()

```
void starpu_bus_set_direct (
             int busid,
             int direct )
```

See Hardware Topology for more details.

57.31.4.11 starpu_bus_get_direct()

```
int starpu_bus_get_direct (
             int busid )
```

See Hardware Topology for more details.

57.31.4.12 starpu_bus_set_ngpus()

```
void starpu_bus_set_ngpus (
             int busid,
             int ngpus )
```

See Hardware Topology for more details.

57.31.4.13 starpu_bus_get_ngpus()

```
int starpu_bus_get_ngpus (
             int busid )
```

See Hardware Topology for more details.

57.31.4.14 starpu_bus_get_profiling_info()

```
int starpu_bus_get_profiling_info (
             int busid,
             struct starpu_profiling_bus_info * bus_info )
```

See _starpu_profiling_bus_helper_display_summary in src/profiling/profiling_helpers.c for a usage example. Note that calling starpu_bus_get_profiling_info() resets the counters to zero. See Feedback Figures for more details.

57.31.4.15 starpu_timing_timespec_delay_us()

```
double starpu_timing_timespec_delay_us (
             struct timespec * start,
             struct timespec * end )
```

Return the time elapsed between `start` and `end` in microseconds. See Per-task Feedback for more details.

57.31.4.16 starpu_timing_timespec_to_us()

```
double starpu_timing_timespec_to_us (
             struct timespec * ts )
```

Convert the given timespec `ts` into microseconds. See Per-task Feedback for more details.

57.31.4.17 starpu_profiling_bus_helper_display_summary()

```
void starpu_profiling_bus_helper_display_summary (
             void )
```

Display statistics about the bus on `stderr`. if the environment variable STARPU_BUS_STATS is defined. The function is called automatically by starpu_shutdown(). See Data Statistics for more details.

57.31.4.18 starpu_profiling_worker_helper_display_summary()

```
void starpu_profiling_worker_helper_display_summary (
             void )
```

Display statistic about the workers on `stderr` if the environment variable STARPU_WORKER_STATS is defined. The function is called automatically by starpu_shutdown(). See Data Statistics for more details.

57.31.4.19 starpu_data_display_memory_stats()

```
void starpu_data_display_memory_stats (
            void  )
```

Display statistics about the current data handles registered within StarPU. StarPU must have been configured with the configure option --enable-memory-stats (see Memory Feedback). See Memory Feedback for more details.

57.32 Profiling Tool

Data Structures

- struct starpu_prof_tool_info
- union starpu_prof_tool_event_info
- struct starpu_prof_tool_api_info

Typedefs

- typedef void(∗ **starpu_prof_tool_cb_func**) (struct starpu_prof_tool_info ∗, union starpu_prof_tool_event_info ∗, struct starpu_prof_tool_api_info ∗)
- typedef void(∗ starpu_prof_tool_entry_register_func) (enum starpu_prof_tool_event event_type, starpu_prof_tool_cb_func cb, enum starpu_prof_tool_command info)
- typedef void(∗ starpu_prof_tool_entry_func) (starpu_prof_tool_entry_register_func reg, starpu_prof_tool_entry_register_func unreg)

Enumerations

- enum starpu_prof_tool_event { **starpu_prof_tool_event_none** , **starpu_prof_tool_event_init** , **starpu_prof_tool_event_terminate** , **starpu_prof_tool_event_init_begin** , **starpu_prof_tool_event_init_end** , **starpu_prof_tool_event_driver_init** , **starpu_prof_tool_event_driver_deinit** , **starpu_prof_tool_event_driver_init_start** , **starpu_prof_tool_event_driver_init_end** , **starpu_prof_tool_event_start_cpu_exec** , **starpu_prof_tool_event_end_cpu_exec** , **starpu_prof_tool_event_start_gpu_exec** , **starpu_prof_tool_event_end_gpu_exec** , **starpu_prof_tool_event_start_transfer** , **starpu_prof_tool_event_end_transfer** , **starpu_prof_tool_event_user_start** , **starpu_prof_tool_event_user_end** }
- enum starpu_prof_tool_driver_type { **starpu_prof_tool_driver_cpu** , **starpu_prof_tool_driver_gpu** , **starpu_prof_tool_driver_hip** , **starpu_prof_tool_driver_ocl** }
- enum starpu_prof_tool_command { **starpu_prof_tool_command_reg** , **starpu_prof_tool_command_toggle** , **starpu_prof_tool_command_toggle_per_thread** }

57.32.1 Detailed Description

57.32.2 Data Structure Documentation

57.32.2.1 struct starpu_prof_tool_info

General information

Data Fields

struct starpu_conf ∗	conf	
enum starpu_prof_tool_event	event_type	
unsigned int	starpu_version[3]	
int	thread_id	
int	worker_id	
int	device_number	
enum starpu_prof_tool_driver_type	driver_type	
unsigned	memnode	
unsigned	bytes_to_transfer	
unsigned	bytes_transfered	
void ∗	fun_ptr	

57.32.2.2 union starpu_prof_tool_event_info

Event info

Data Fields

enum starpu_prof_tool_event	event_type	

57.32.2.3 struct starpu_prof_tool_api_info

API info

57.32.3 Typedef Documentation

57.32.3.1 starpu_prof_tool_entry_register_func

```
typedef void(* starpu_prof_tool_entry_register_func) (enum starpu_prof_tool_event event_type,
starpu_prof_tool_cb_func cb, enum starpu_prof_tool_command info)
```

Register / unregister events

57.32.3.2 starpu_prof_tool_entry_func

```
typedef void(* starpu_prof_tool_entry_func) (starpu_prof_tool_entry_register_func reg, starpu_prof_tool_entry_
unreg)
```

A function with this signature must be implemented by external tools that want to use the callbacks

57.32.4 Enumeration Type Documentation

57.32.4.1 starpu_prof_tool_event

`enum starpu_prof_tool_event`

Event type

57.32.4.2 starpu_prof_tool_driver_type

`enum starpu_prof_tool_driver_type`

todo

57.32.4.3 starpu_prof_tool_command

`enum starpu_prof_tool_command`

todo

57.33 Random Functions

Macros

- #define **starpu_seed**(seed)
- #define **starpu_srand48**(seed)
- #define **starpu_drand48**()
- #define **starpu_lrand48**()
- #define **starpu_erand48**(xsubi)
- #define **starpu_srand48_r**(seed, buffer)
- #define **starpu_erand48_r**(xsubi, buffer, result)

Typedefs

- typedef int **starpu_drand48_data**

57.33.1 Detailed Description

57.34 Running Drivers

Data Structures

- struct starpu_driver
- union starpu_driver.id

Functions

- void starpu_drivers_preinit (void)
- int starpu_driver_run (struct starpu_driver ∗d)
- void starpu_drivers_request_termination (void)
- int starpu_driver_init (struct starpu_driver ∗d)
- int starpu_driver_run_once (struct starpu_driver ∗d)
- int starpu_driver_deinit (struct starpu_driver ∗d)

57.34.1 Detailed Description

57.34.2 Data Structure Documentation

57.34.2.1 struct starpu_driver

structure for designating a given driver. See Using The Driver API for more details.

Data Fields

enum starpu_worker_archtype	type	Type of the driver. Only STARPU_CPU_WORKER, STARPU_CUDA_WORKER and STARPU_OPENCL_WORKER are currently supported.
union starpu_driver.id	id	Identifier of the driver.

57.34.2.2 union starpu_driver.id

Identifier of the driver.

Data Fields

unsigned	cpu_id	
unsigned	cuda_id	
unsigned	hip_id	
cl_device_id	opencl_id	

57.34.3 Function Documentation

57.34.3.1 starpu_drivers_preinit()

```
void starpu_drivers_preinit (
            void )
```

Pre-initialize drivers So as to register information on device types, memory types, etc. Only use internally by StarPU.

57.34.3.2 starpu_driver_run()

```
int starpu_driver_run (
                struct starpu_driver * d )
```

Initialize the given driver, run it until it receives a request to terminate, deinitialize it and return 0 on success. Return `-EINVAL` if starpu_driver::type is not a valid StarPU device type (STARPU_CPU_WORKER, STARPU_CUDA_WORKER or STARPU_OPENCL_WORKER).
This is the same as using the following functions: calling starpu_driver_init(), then calling starpu_driver_run_once() in a loop, and finally starpu_driver_deinit().
See Using The Driver API for more details.

57.34.3.3 starpu_drivers_request_termination()

```
void starpu_drivers_request_termination (
                void )
```

Notify all running drivers that they should terminate. See Using The Driver API for more details.

57.34.3.4 starpu_driver_init()

```
int starpu_driver_init (
                struct starpu_driver * d )
```

Initialize the given driver. Return 0 on success, `-EINVAL` if starpu_driver::type is not a valid starpu_worker_archtype.
See Using The Driver API for more details.

57.34.3.5 starpu_driver_run_once()

```
int starpu_driver_run_once (
                struct starpu_driver * d )
```

Run the driver once, then return 0 on success, `-EINVAL` if starpu_driver::type is not a valid starpu_worker_archtype.
See Using The Driver API for more details.

57.34.3.6 starpu_driver_deinit()

```
int starpu_driver_deinit (
                struct starpu_driver * d )
```

Deinitialize the given driver. Return 0 on success, `-EINVAL` if starpu_driver::type is not a valid starpu_worker_archtype.
See Using The Driver API for more details.

57.35 Scheduler Toolbox

This is the interface for the scheduler toolbox.

Typedefs

- typedef struct starpu_st_fifo_taskq ∗ starpu_st_fifo_taskq_t
- typedef struct starpu_st_prio_deque ∗ starpu_st_prio_deque_t

Functions

- starpu_st_fifo_taskq_t starpu_st_fifo_taskq_create (void) STARPU_ATTRIBUTE_MALLOC
- void **starpu_st_fifo_taskq_init** (starpu_st_fifo_taskq_t fifo)
- void **starpu_st_fifo_taskq_destroy** (starpu_st_fifo_taskq_t fifo)
- int **starpu_st_fifo_taskq_empty** (starpu_st_fifo_taskq_t fifo)
- double **starpu_st_fifo_taskq_get_exp_len_prev_task_list** (starpu_st_fifo_taskq_t fifo_queue, struct starpu_task ∗task, int workerid, int nimpl, int ∗fifo_ntasks)
- unsigned starpu_st_fifo_ntasks_get (starpu_st_fifo_taskq_t fifo)
- void starpu_st_fifo_ntasks_inc (starpu_st_fifo_taskq_t fifo, int n)
- unsigned ∗ starpu_st_fifo_ntasks_per_priority_get (starpu_st_fifo_taskq_t fifo)
- unsigned starpu_st_fifo_nprocessed_get (starpu_st_fifo_taskq_t fifo)
- void starpu_st_fifo_nprocessed_inc (starpu_st_fifo_taskq_t fifo, int n)
- double starpu_st_fifo_exp_start_get (starpu_st_fifo_taskq_t fifo)
- void starpu_st_fifo_exp_start_set (starpu_st_fifo_taskq_t fifo, double exp_start)
- double starpu_st_fifo_exp_end_get (starpu_st_fifo_taskq_t fifo)
- void starpu_st_fifo_exp_end_set (starpu_st_fifo_taskq_t fifo, double exp_end)
- double starpu_st_fifo_exp_len_get (starpu_st_fifo_taskq_t fifo)
- void starpu_st_fifo_exp_len_set (starpu_st_fifo_taskq_t fifo, double exp_len)
- void starpu_st_fifo_exp_len_inc (starpu_st_fifo_taskq_t fifo, double exp_len)
- double ∗ starpu_st_fifo_exp_len_per_priority_get (starpu_st_fifo_taskq_t fifo)
- double starpu_st_fifo_pipeline_len_get (starpu_st_fifo_taskq_t fifo)
- void starpu_st_fifo_pipeline_len_set (starpu_st_fifo_taskq_t fifo, double pipeline_len)
- void starpu_st_fifo_pipeline_len_inc (starpu_st_fifo_taskq_t fifo, double pipeline_len)
- int **starpu_st_fifo_taskq_push_sorted_task** (starpu_st_fifo_taskq_t fifo_queue, struct starpu_task ∗task)
- int **starpu_st_fifo_taskq_push_task** (starpu_st_fifo_taskq_t fifo, struct starpu_task ∗task)
- int **starpu_st_fifo_taskq_push_back_task** (starpu_st_fifo_taskq_t fifo_queue, struct starpu_task ∗task)
- int **starpu_st_fifo_taskq_pop_this_task** (starpu_st_fifo_taskq_t fifo_queue, int workerid, struct starpu_task ∗task)
- struct starpu_task ∗ **starpu_st_fifo_taskq_pop_task** (starpu_st_fifo_taskq_t fifo, int workerid)
- struct starpu_task ∗ starpu_st_fifo_taskq_pop_local_task (starpu_st_fifo_taskq_t fifo)
- struct starpu_task ∗ starpu_st_fifo_taskq_pop_first_ready_task (starpu_st_fifo_taskq_t fifo_queue, unsigned workerid, int num_priorities)
- void starpu_st_prio_deque_init (starpu_st_prio_deque_t pdeque)
- void **starpu_st_prio_deque_destroy** (starpu_st_prio_deque_t pdeque)
- int starpu_st_prio_deque_is_empty (starpu_st_prio_deque_t pdeque)
- int **starpu_st_prio_deque_push_back_task** (starpu_st_prio_deque_t pdeque, struct starpu_task ∗task)
- int starpu_st_prio_deque_push_front_task (starpu_st_prio_deque_t pdeque, struct starpu_task ∗task)
- struct starpu_task ∗ starpu_st_prio_deque_pop_task_for_worker (starpu_st_prio_deque_t pdeque, int workerid, struct starpu_task ∗∗skipped)
- struct starpu_task ∗ starpu_st_prio_deque_deque_task_for_worker (starpu_st_prio_deque_t pdeque, int workerid, struct starpu_task ∗∗skipped)
- struct starpu_task ∗ **starpu_st_prio_deque_deque_first_ready_task** (starpu_st_prio_deque_t pdeque, unsigned workerid)
- struct starpu_task ∗ **starpu_st_prio_deque_pop_task** (starpu_st_prio_deque_t pdeque)
- struct starpu_task ∗ **starpu_st_prio_deque_highest_task** (starpu_st_prio_deque_t pdeque)
- struct starpu_task ∗ **starpu_st_prio_deque_pop_back_task** (starpu_st_prio_deque_t pdeque)

- int **starpu_st_prio_deque_pop_this_task** (starpu_st_prio_deque_t pdeque, int workerid, struct starpu_task ∗task)
- void **starpu_st_prio_deque_erase** (starpu_st_prio_deque_t pdeque, struct starpu_task ∗task)
- int **starpu_st_normalize_prio** (int priority, int num_priorities, unsigned sched_ctx_id)
- int **starpu_st_non_ready_buffers_count** (struct starpu_task ∗task, unsigned worker)
- void **starpu_st_non_ready_buffers_size** (struct starpu_task ∗task, unsigned worker, size_t ∗non_readyp, size_t ∗non_loadingp, size_t ∗non_allocatedp)

57.35.1 Detailed Description

This is the interface for the scheduler toolbox.
The definitions of the different queue types below (e.g starpu_st_fifo_taskq_t) are private and are thus not available outside the StarPU source directory. Hence when defining your own scheduler outside of StarPU source directory, you should use the functions below. Look for example in the scheduler defined in `examples/cholesky/libmy_dmda.c`

57.35.2 Typedef Documentation

57.35.2.1 starpu_st_fifo_taskq_t

```
typedef struct starpu_st_fifo_taskq* starpu_st_fifo_taskq_t
```

Opaque type for FIFO task queue

57.35.2.2 starpu_st_prio_deque_t

```
typedef struct starpu_st_prio_deque* starpu_st_prio_deque_t
```

Opaque type for PRIO task queue

57.35.3 Function Documentation

57.35.3.1 starpu_st_fifo_taskq_create()

```
starpu_st_fifo_taskq_t starpu_st_fifo_taskq_create (
            void )
```

Create a FIFO task queue

57.35.3.2 starpu_st_fifo_ntasks_get()

```
unsigned starpu_st_fifo_ntasks_get (
            starpu_st_fifo_taskq_t fifo )
```

get the number of tasks currently in the queue

57.35.3.3 starpu_st_fifo_ntasks_inc()

```
void starpu_st_fifo_ntasks_inc (
            starpu_st_fifo_taskq_t fifo,
            int n )
```

increase by n the number of tasks currently in the queue

57.35.3.4 starpu_st_fifo_ntasks_per_priority_get()

```
unsigned * starpu_st_fifo_ntasks_per_priority_get (
            starpu_st_fifo_taskq_t fifo )
```

get the number of tasks currently in the queue corresponding to each priority

57.35.3.5 starpu_st_fifo_nprocessed_get()

```
unsigned starpu_st_fifo_nprocessed_get (
            starpu_st_fifo_taskq_t fifo )
```

get the number of tasks that were processed

57.35.3.6 starpu_st_fifo_nprocessed_inc()

```
void starpu_st_fifo_nprocessed_inc (
            starpu_st_fifo_taskq_t fifo,
            int n )
```

increase by n the number of tasks that were processed

57.35.3.7 starpu_st_fifo_exp_start_get()

```
double starpu_st_fifo_exp_start_get (
            starpu_st_fifo_taskq_t fifo )
```

only meaningful if the queue is only used by a single worker Get the expected start date of next item to do in the queue (i.e. not started yet). This is thus updated when we start it.

57.35.3.8 starpu_st_fifo_exp_start_set()

```
void starpu_st_fifo_exp_start_set (
            starpu_st_fifo_taskq_t fifo,
            double exp_start )
```

Set the expected start date of next item to do in the queue (i.e. not started yet).

57.35.3.9 starpu_st_fifo_exp_end_get()

```
double starpu_st_fifo_exp_end_get (
            starpu_st_fifo_taskq_t fifo )
```

get the expected end date of last task in the queue

57.35.3.10 starpu_st_fifo_exp_end_set()

```
void starpu_st_fifo_exp_end_set (
            starpu_st_fifo_taskq_t fifo,
            double exp_end )
```

set the expected end date of last task in the queue

57.35.3.11 starpu_st_fifo_exp_len_get()

```
double starpu_st_fifo_exp_len_get (
            starpu_st_fifo_taskq_t fifo )
```

get the expected duration of the set of tasks in the queue

57.35.3.12 starpu_st_fifo_exp_len_set()

```
void starpu_st_fifo_exp_len_set (
            starpu_st_fifo_taskq_t fifo,
            double exp_len )
```

set the expected duration of the set of tasks in the queue

57.35.3.13 starpu_st_fifo_exp_len_inc()

```
void starpu_st_fifo_exp_len_inc (
            starpu_st_fifo_taskq_t fifo,
            double exp_len )
```

increase or decrease the expected duration of the set of tasks in the queue

57.35.3.14 starpu_st_fifo_exp_len_per_priority_get()

```
double * starpu_st_fifo_exp_len_per_priority_get (
            starpu_st_fifo_taskq_t fifo )
```

get the expected duration of the set of tasks in the queue corresponding to each priority

57.35.3.15 starpu_st_fifo_pipeline_len_get()

```
double starpu_st_fifo_pipeline_len_get (
            starpu_st_fifo_taskq_t fifo )
```

get the expected duration of what is already pushed to the worker

57.35.3.16 starpu_st_fifo_pipeline_len_set()

```
void starpu_st_fifo_pipeline_len_set (
            starpu_st_fifo_taskq_t fifo,
            double pipeline_len )
```

set the expected duration of what is already pushed to the worker

57.35.3.17 starpu_st_fifo_pipeline_len_inc()

```
void starpu_st_fifo_pipeline_len_inc (
            starpu_st_fifo_taskq_t fifo,
            double pipeline_len )
```

increase the expected duration of what is already pushed to the worker (the value can be negative)

57.35.3.18 starpu_st_fifo_taskq_pop_local_task()

```
struct starpu_task * starpu_st_fifo_taskq_pop_local_task (
            starpu_st_fifo_taskq_t fifo )
```

This is the same as starpu_st_fifo_taskq_pop_task(), but without checking that the worker will be able to execute this task. This is useful when the scheduler has already checked it.

57.35.3.19 starpu_st_fifo_taskq_pop_first_ready_task()

```
struct starpu_task * starpu_st_fifo_taskq_pop_first_ready_task (
            starpu_st_fifo_taskq_t fifo_queue,
            unsigned workerid,
            int num_priorities )
```

Pop the first task that can be executed on the calling driver and taking into account readiness of data

57.35.3.20 starpu_st_prio_deque_init()

```
void starpu_st_prio_deque_init (
            starpu_st_prio_deque_t pdeque )
```

all _starpu_prio_deque_pop/deque_task function return a task or a NULL pointer if none are available in O(lg(nb priorities))

57.35.3.21 starpu_st_prio_deque_is_empty()

```
int starpu_st_prio_deque_is_empty (
            starpu_st_prio_deque_t pdeque )
```

return 0 iff the struct starpu_st_prio_deque is not empty

57.35.3.22 starpu_st_prio_deque_push_front_task()

```
int starpu_st_prio_deque_push_front_task (
            starpu_st_prio_deque_t pdeque,
            struct starpu_task * task )
```

push a task in O(lg(nb priorities))

57.35.3.23 starpu_st_prio_deque_pop_task_for_worker()

```
struct starpu_task * starpu_st_prio_deque_pop_task_for_worker (
            starpu_st_prio_deque_t pdeque,
            int workerid,
            struct starpu_task ** skipped )
```

deque a task of the higher priority available from the front of the list for the highest priority

57.35.3.24 starpu_st_prio_deque_deque_task_for_worker()

```
struct starpu_task * starpu_st_prio_deque_deque_task_for_worker (
            starpu_st_prio_deque_t pdeque,
            int workerid,
            struct starpu_task ** skipped )
```

return a task that can be executed by workerid from the back of the list for the highest priority

57.36 Scheduling Contexts

StarPU permits on one hand grouping workers in combined workers in order to execute a parallel task and on the other hand grouping tasks in bundles that will be executed by a single specified worker. In contrast when we group workers in scheduling contexts we submit starpu tasks to them and we schedule them with the policy assigned to the context. Scheduling contexts can be created, deleted and modified dynamically.

Scheduling Contexts Basic API

- void(∗)(unsigned) starpu_sched_ctx_get_sched_policy_callback (unsigned sched_ctx_id)
- unsigned starpu_sched_ctx_create (int ∗workerids_ctx, int nworkers_ctx, const char ∗sched_ctx_name,...)
- unsigned starpu_sched_ctx_create_inside_interval (const char ∗policy_name, const char ∗sched_ctx_name, int min_ncpus, int max_ncpus, int min_ngpus, int max_ngpus, unsigned allow_overlap)
- void starpu_sched_ctx_register_close_callback (unsigned sched_ctx_id, void(∗close_callback)(unsigned sched_ctx_id, void ∗args), void ∗args)
- void starpu_sched_ctx_add_workers (int ∗workerids_ctx, unsigned nworkers_ctx, unsigned sched_ctx_id)
- void starpu_sched_ctx_remove_workers (int ∗workerids_ctx, unsigned nworkers_ctx, unsigned sched_ctx_id)
- void starpu_sched_ctx_display_workers (unsigned sched_ctx_id, FILE ∗f)
- void starpu_sched_ctx_delete (unsigned sched_ctx_id)
- void starpu_sched_ctx_set_inheritor (unsigned sched_ctx_id, unsigned inheritor)
- unsigned **starpu_sched_ctx_get_inheritor** (unsigned sched_ctx_id)
- unsigned **starpu_sched_ctx_get_hierarchy_level** (unsigned sched_ctx_id)
- void starpu_sched_ctx_set_context (unsigned ∗sched_ctx_id)
- unsigned starpu_sched_ctx_get_context (void)
- void starpu_sched_ctx_stop_task_submission (void)
- void starpu_sched_ctx_finished_submit (unsigned sched_ctx_id)
- unsigned starpu_sched_ctx_get_workers_list (unsigned sched_ctx_id, int ∗∗workerids)
- unsigned starpu_sched_ctx_get_workers_list_raw (unsigned sched_ctx_id, int ∗∗workerids)
- unsigned starpu_sched_ctx_get_nworkers (unsigned sched_ctx_id)
- unsigned starpu_sched_ctx_get_nshared_workers (unsigned sched_ctx_id, unsigned sched_ctx_id2)
- unsigned starpu_sched_ctx_contains_worker (int workerid, unsigned sched_ctx_id)
- unsigned **starpu_sched_ctx_contains_type_of_worker** (enum starpu_worker_archtype arch, unsigned sched_ctx_id)
- unsigned starpu_sched_ctx_worker_get_id (unsigned sched_ctx_id)
- unsigned **starpu_sched_ctx_get_ctx_for_task** (struct starpu_task ∗task)
- unsigned **starpu_worker_get_sched_ctx_id_stream** (unsigned stream_workerid)
- unsigned starpu_sched_ctx_overlapping_ctxs_on_worker (int workerid)
- void ∗ starpu_sched_ctx_get_user_data (unsigned sched_ctx_id)
- void **starpu_sched_ctx_set_user_data** (unsigned sched_ctx_id, void ∗user_data)
- void starpu_sched_ctx_set_policy_data (unsigned sched_ctx_id, void ∗policy_data)
- void ∗ starpu_sched_ctx_get_policy_data (unsigned sched_ctx_id)
- struct starpu_sched_policy ∗ **starpu_sched_ctx_get_sched_policy** (unsigned sched_ctx_id)
- void ∗ starpu_sched_ctx_exec_parallel_code (void ∗(∗func)(void ∗), void ∗param, unsigned sched_ctx_id)
- int **starpu_sched_ctx_get_nready_tasks** (unsigned sched_ctx_id)
- double **starpu_sched_ctx_get_nready_flops** (unsigned sched_ctx_id)
- void **starpu_sched_ctx_list_task_counters_increment** (unsigned sched_ctx_id, int workerid)
- void **starpu_sched_ctx_list_task_counters_decrement** (unsigned sched_ctx_id, int workerid)
- void **starpu_sched_ctx_list_task_counters_reset** (unsigned sched_ctx_id, int workerid)
- void **starpu_sched_ctx_list_task_counters_increment_all_ctx_locked** (struct starpu_task ∗task, unsigned sched_ctx_id)
- void **starpu_sched_ctx_list_task_counters_decrement_all_ctx_locked** (struct starpu_task ∗task, unsigned sched_ctx_id)
- void **starpu_sched_ctx_list_task_counters_reset_all** (struct starpu_task ∗task, unsigned sched_ctx_id)
- void **starpu_sched_ctx_set_priority** (int ∗workers, int nworkers, unsigned sched_ctx_id, unsigned priority)
- unsigned **starpu_sched_ctx_get_priority** (int worker, unsigned sched_ctx_id)

- void **starpu_sched_ctx_get_available_cpuids** (unsigned sched_ctx_id, int ∗∗cpuids, int ∗ncpuids)
- void **starpu_sched_ctx_bind_current_thread_to_cpuid** (unsigned cpuid)
- int **starpu_sched_ctx_book_workers_for_task** (unsigned sched_ctx_id, int ∗workerids, int nworkers)
- void **starpu_sched_ctx_unbook_workers_for_task** (unsigned sched_ctx_id, int master)
- unsigned starpu_sched_ctx_worker_is_master_for_child_ctx (int workerid, unsigned sched_ctx_id)
- unsigned starpu_sched_ctx_master_get_context (int masterid)
- void **starpu_sched_ctx_revert_task_counters_ctx_locked** (unsigned sched_ctx_id, double flops)
- void **starpu_sched_ctx_move_task_to_ctx_locked** (struct starpu_task ∗task, unsigned sched_ctx, unsigned with_repush)
- int **starpu_sched_ctx_get_worker_rank** (unsigned sched_ctx_id)
- unsigned **starpu_sched_ctx_has_starpu_scheduler** (unsigned sched_ctx_id, unsigned ∗awake_workers)
- int **starpu_sched_ctx_get_stream_worker** (unsigned sub_ctx)
- int **starpu_sched_ctx_get_nsms** (unsigned sched_ctx)
- void **starpu_sched_ctx_get_sms_interval** (int stream_workerid, int ∗start, int ∗end)
- #define STARPU_SCHED_CTX_POLICY_NAME
- #define STARPU_SCHED_CTX_POLICY_STRUCT
- #define STARPU_SCHED_CTX_POLICY_MIN_PRIO
- #define STARPU_SCHED_CTX_POLICY_MAX_PRIO
- #define **STARPU_SCHED_CTX_HIERARCHY_LEVEL**
- #define **STARPU_SCHED_CTX_NESTED**
- #define STARPU_SCHED_CTX_AWAKE_WORKERS
- #define STARPU_SCHED_CTX_POLICY_INIT
- #define STARPU_SCHED_CTX_USER_DATA
- #define STARPU_SCHED_CTX_CUDA_NSMS
- #define STARPU_SCHED_CTX_SUB_CTXS

Scheduling Context Priorities

- int starpu_sched_ctx_get_min_priority (unsigned sched_ctx_id)
- int starpu_sched_ctx_get_max_priority (unsigned sched_ctx_id)
- int starpu_sched_ctx_set_min_priority (unsigned sched_ctx_id, int min_prio)
- int starpu_sched_ctx_set_max_priority (unsigned sched_ctx_id, int max_prio)
- int **starpu_sched_ctx_min_priority_is_set** (unsigned sched_ctx_id)
- int **starpu_sched_ctx_max_priority_is_set** (unsigned sched_ctx_id)
- #define STARPU_MIN_PRIO
- #define STARPU_MAX_PRIO
- #define STARPU_DEFAULT_PRIO

Scheduling Context Worker Collection

- struct starpu_worker_collection ∗ starpu_sched_ctx_create_worker_collection (unsigned sched_ctx_id, enum starpu_worker_collection_type type) STARPU_ATTRIBUTE_MALLOC
- void starpu_sched_ctx_delete_worker_collection (unsigned sched_ctx_id)
- struct starpu_worker_collection ∗ starpu_sched_ctx_get_worker_collection (unsigned sched_ctx_id)

57.36.1 Detailed Description

StarPU permits on one hand grouping workers in combined workers in order to execute a parallel task and on the other hand grouping tasks in bundles that will be executed by a single specified worker. In contrast when we group workers in scheduling contexts we submit starpu tasks to them and we schedule them with the policy assigned to the context. Scheduling contexts can be created, deleted and modified dynamically.

57.36.2 Macro Definition Documentation

57.36.2.1 STARPU_SCHED_CTX_POLICY_NAME

`#define STARPU_SCHED_CTX_POLICY_NAME`
Used when calling starpu_sched_ctx_create() to specify a name for a scheduling policy

57.36.2.2 STARPU_SCHED_CTX_POLICY_STRUCT

`#define STARPU_SCHED_CTX_POLICY_STRUCT`
Used when calling starpu_sched_ctx_create() to specify a pointer to a scheduling policy

57.36.2.3 STARPU_SCHED_CTX_POLICY_MIN_PRIO

`#define STARPU_SCHED_CTX_POLICY_MIN_PRIO`
Used when calling starpu_sched_ctx_create() to specify a minimum scheduler priority value.

57.36.2.4 STARPU_SCHED_CTX_POLICY_MAX_PRIO

`#define STARPU_SCHED_CTX_POLICY_MAX_PRIO`
Used when calling starpu_sched_ctx_create() to specify a maximum scheduler priority value.

57.36.2.5 STARPU_SCHED_CTX_AWAKE_WORKERS

`#define STARPU_SCHED_CTX_AWAKE_WORKERS`
Used when calling starpu_sched_ctx_create() to specify ???

57.36.2.6 STARPU_SCHED_CTX_POLICY_INIT

`#define STARPU_SCHED_CTX_POLICY_INIT`
Used when calling starpu_sched_ctx_create() to specify a function pointer allowing to initialize the scheduling policy.

57.36.2.7 STARPU_SCHED_CTX_USER_DATA

`#define STARPU_SCHED_CTX_USER_DATA`
Used when calling starpu_sched_ctx_create() to specify a pointer to some user data related to the context being created.

57.36.2.8 STARPU_SCHED_CTX_CUDA_NSMS

`#define STARPU_SCHED_CTX_CUDA_NSMS`
Used when calling starpu_sched_ctx_create() in order to create a context on the NVIDIA GPU to specify the number of SMs the context should have

57.36.2.9 STARPU_SCHED_CTX_SUB_CTXS

`#define STARPU_SCHED_CTX_SUB_CTXS`
Used when calling starpu_sched_ctx_create() to specify a list of sub contexts of the current context.

57.36.2.10 STARPU_MIN_PRIO

`#define STARPU_MIN_PRIO`
Provided for legacy reasons.

57.36.2.11 STARPU_MAX_PRIO

`#define STARPU_MAX_PRIO`
Provided for legacy reasons.

57.36.2.12 STARPU_DEFAULT_PRIO

`#define STARPU_DEFAULT_PRIO`

By convention, the default priority level should be 0 so that we can statically allocate tasks with a default priority.

57.36.3 Function Documentation

57.36.3.1 starpu_sched_ctx_create()

```
unsigned starpu_sched_ctx_create (
              int * workerids_ctx,
              int nworkers_ctx,
              const char * sched_ctx_name,
               ... )
```

Create a scheduling context with the given parameters (see below) and assign the workers in `workerids_ctx` to execute the tasks submitted to it. The return value represents the identifier of the context that has just been created. It will be further used to indicate the context the tasks will be submitted to. The return value should be at most STARPU_NMAX_SCHED_CTXS.

The arguments following the name of the scheduling context can be of the following types:

- STARPU_SCHED_CTX_POLICY_NAME, followed by the name of a predefined scheduling policy. Use an empty string to create the context with the default scheduling policy.
- STARPU_SCHED_CTX_POLICY_STRUCT, followed by a pointer to a custom scheduling policy (struct starpu_sched_policy ∗)
- STARPU_SCHED_CTX_POLICY_MIN_PRIO, followed by a integer representing the minimum priority value to be defined for the scheduling policy.
- STARPU_SCHED_CTX_POLICY_MAX_PRIO, followed by a integer representing the maximum priority value to be defined for the scheduling policy.
- STARPU_SCHED_CTX_POLICY_INIT, followed by a function pointer (ie. void init_sched(void)) allowing to initialize the scheduling policy.
- STARPU_SCHED_CTX_USER_DATA, followed by a pointer to a custom user data structure, to be retrieved by starpu_sched_ctx_get_user_data().

See Creating A Context for more details.

57.36.3.2 starpu_sched_ctx_create_inside_interval()

```
unsigned starpu_sched_ctx_create_inside_interval (
              const char * policy_name,
              const char * sched_ctx_name,
              int min_ncpus,
              int max_ncpus,
              int min_ngpus,
              int max_ngpus,
              unsigned allow_overlap )
```

Create a context indicating an approximate interval of resources

57.36.3.3 starpu_sched_ctx_register_close_callback()

```
void starpu_sched_ctx_register_close_callback (
              unsigned sched_ctx_id,
              void(*)(unsigned sched_ctx_id, void *args) close_callback,
              void * args )
```

Execute the callback whenever the last task of the context finished executing, it is called with the parameters `sched_ctx` and any other parameter needed by the application (packed in `args`)

57.36.3.4 starpu_sched_ctx_add_workers()

```
void starpu_sched_ctx_add_workers (
            int * workerids_ctx,
            unsigned nworkers_ctx,
            unsigned sched_ctx_id )
```

Add dynamically the workers in `workerids_ctx` to the context `sched_ctx_id`. The last argument cannot be greater than STARPU_NMAX_SCHED_CTXS. See Modifying A Context for more details.

57.36.3.5 starpu_sched_ctx_remove_workers()

```
void starpu_sched_ctx_remove_workers (
            int * workerids_ctx,
            unsigned nworkers_ctx,
            unsigned sched_ctx_id )
```

Remove the workers in `workerids_ctx` from the context `sched_ctx_id`. The last argument cannot be greater than STARPU_NMAX_SCHED_CTXS. See Modifying A Context for more details.

57.36.3.6 starpu_sched_ctx_display_workers()

```
void starpu_sched_ctx_display_workers (
            unsigned sched_ctx_id,
            FILE * f )
```

Print on the file `f` the worker names belonging to the context `sched_ctx_id`

57.36.3.7 starpu_sched_ctx_delete()

```
void starpu_sched_ctx_delete (
            unsigned sched_ctx_id )
```

Delete scheduling context `sched_ctx_id` and transfer remaining workers to the inheritor scheduling context. See Deleting A Context for more details.

57.36.3.8 starpu_sched_ctx_set_inheritor()

```
void starpu_sched_ctx_set_inheritor (
            unsigned sched_ctx_id,
            unsigned inheritor )
```

Indicate that the context `inheritor` will inherit the resources of the context `sched_ctx_id` when `sched_ctx_id` will be deleted. See Deleting A Context for more details.

57.36.3.9 starpu_sched_ctx_set_context()

```
void starpu_sched_ctx_set_context (
            unsigned * sched_ctx_id )
```

Set the scheduling context the subsequent tasks will be submitted to. See Submitting Tasks To A Context and Temporary Contexts for more details.

57.36.3.10 starpu_sched_ctx_get_context()

```
unsigned starpu_sched_ctx_get_context (
            void )
```

Return the scheduling context the tasks are currently submitted to, or STARPU_NMAX_SCHED_CTXS if no default context has been defined by calling the function starpu_sched_ctx_set_context().

57.36.3.11 starpu_sched_ctx_stop_task_submission()

```
void starpu_sched_ctx_stop_task_submission (
            void )
```

Stop submitting tasks from the empty context list until the next time the context has time to check the empty context list. See Emptying A Context for more details.

57.36.3.12 starpu_sched_ctx_finished_submit()

```
void starpu_sched_ctx_finished_submit (
            unsigned sched_ctx_id )
```

Indicate starpu that the application finished submitting to this context in order to move the workers to the inheritor as soon as possible. See Deleting A Context for more details.

57.36.3.13 starpu_sched_ctx_get_workers_list()

```
unsigned starpu_sched_ctx_get_workers_list (
            unsigned sched_ctx_id,
            int ** workerids )
```

Return the list of workers in the array `workerids`, the return value is the number of workers. The user should free the `workerids` table after finishing using it (it is allocated inside the function with the proper size)

57.36.3.14 starpu_sched_ctx_get_workers_list_raw()

```
unsigned starpu_sched_ctx_get_workers_list_raw (
            unsigned sched_ctx_id,
            int ** workerids )
```

Return the list of workers in the array `workerids`, the return value is the number of workers. This list is provided in raw order, i.e. not sorted by tree or list order, and the user should not free the `workerids` table. This function is thus much less costly than starpu_sched_ctx_get_workers_list().

57.36.3.15 starpu_sched_ctx_get_nworkers()

```
unsigned starpu_sched_ctx_get_nworkers (
            unsigned sched_ctx_id )
```

Return the number of workers managed by the specified context (Usually needed to verify if it manages any workers or if it should be blocked)

57.36.3.16 starpu_sched_ctx_get_nshared_workers()

```
unsigned starpu_sched_ctx_get_nshared_workers (
            unsigned sched_ctx_id,
            unsigned sched_ctx_id2 )
```

Return the number of workers shared by two contexts.

57.36.3.17 starpu_sched_ctx_contains_worker()

```
unsigned starpu_sched_ctx_contains_worker (
            int workerid,
            unsigned sched_ctx_id )
```

Return 1 if the worker belongs to the context and 0 otherwise

57.36.3.18 starpu_sched_ctx_worker_get_id()

```
unsigned starpu_sched_ctx_worker_get_id (
            unsigned sched_ctx_id )
```

Return the workerid if the worker belongs to the context and -1 otherwise. If the thread calling this function is not a worker the function returns -1 as it calls the function starpu_worker_get_id().

57.36.3.19 starpu_sched_ctx_overlapping_ctxs_on_worker()

```
unsigned starpu_sched_ctx_overlapping_ctxs_on_worker (
            int workerid )
```

Check if a worker is shared between several contexts

57.36.3.20 starpu_sched_ctx_get_user_data()

```
void * starpu_sched_ctx_get_user_data (
            unsigned sched_ctx_id )
```

Return the user data pointer associated to the scheduling context.

57.36.3.21 starpu_sched_ctx_set_policy_data()

```
void starpu_sched_ctx_set_policy_data (
            unsigned sched_ctx_id,
            void * policy_data )
```

Allocate the scheduling policy data (private information of the scheduler like queues, variables, additional condition variables) the context. See Defining A New Basic Scheduling Policy for more details.

57.36.3.22 starpu_sched_ctx_get_policy_data()

```
void * starpu_sched_ctx_get_policy_data (
            unsigned sched_ctx_id )
```

Return the scheduling policy data (private information of the scheduler) of the contexts previously assigned to. See Defining A New Basic Scheduling Policy for more details.

57.36.3.23 starpu_sched_ctx_exec_parallel_code()

```
void * starpu_sched_ctx_exec_parallel_code (
            void *(*)(void *) func,
            void * param,
            unsigned sched_ctx_id )
```

Execute any parallel code on the workers of the sched_ctx (workers are blocked)

57.36.3.24 starpu_sched_ctx_worker_is_master_for_child_ctx()

```
unsigned starpu_sched_ctx_worker_is_master_for_child_ctx (
            int workerid,
            unsigned sched_ctx_id )
```

Return the first context (child of sched_ctx_id) where the workerid is master

57.36.3.25 starpu_sched_ctx_master_get_context()

```
unsigned starpu_sched_ctx_master_get_context (
            int masterid )
```

Return the context id of masterid if it master of a context. If not, return STARPU_NMAX_SCHED_CTXS.

57.36.3.26 starpu_sched_ctx_get_min_priority()

```
int starpu_sched_ctx_get_min_priority (
            unsigned sched_ctx_id )
```

Return the current minimum priority level supported by the scheduling policy of the given scheduler context.

57.36.3.27 starpu_sched_ctx_get_max_priority()

```
int starpu_sched_ctx_get_max_priority (
            unsigned sched_ctx_id )
```

Return the current maximum priority level supported by the scheduling policy of the given scheduler context.

57.36.3.28 starpu_sched_ctx_set_min_priority()

```
int starpu_sched_ctx_set_min_priority (
            unsigned sched_ctx_id,
            int min_prio )
```

Define the minimum task priority level supported by the scheduling policy of the given scheduler context. The default minimum priority level is the same as the default priority level which is 0 by convention. The application may access that value by calling the function starpu_sched_ctx_get_min_priority(). This function should only be called from the initialization method of the scheduling policy, and should not be used directly from the application.

57.36.3.29 starpu_sched_ctx_set_max_priority()

```
int starpu_sched_ctx_set_max_priority (
            unsigned sched_ctx_id,
            int max_prio )
```

Define the maximum priority level supported by the scheduling policy of the given scheduler context. The default maximum priority level is 1. The application may access that value by calling the starpu_sched_ctx_get_max_priority() function. This function should only be called from the initialization method of the scheduling policy, and should not be used directly from the application.

57.36.3.30 starpu_sched_ctx_create_worker_collection()

```
struct starpu_worker_collection * starpu_sched_ctx_create_worker_collection (
            unsigned sched_ctx_id,
            enum starpu_worker_collection_type type )
```

Create a worker collection of the type indicated by the last parameter for the context specified through the first parameter.

57.36.3.31 starpu_sched_ctx_delete_worker_collection()

```
void starpu_sched_ctx_delete_worker_collection (
            unsigned sched_ctx_id )
```

Delete the worker collection of the specified scheduling context

57.36.3.32 starpu_sched_ctx_get_worker_collection()

```
struct starpu_worker_collection * starpu_sched_ctx_get_worker_collection (
            unsigned sched_ctx_id )
```

Return the worker collection managed by the indicated context

57.36.4 Variable Documentation

57.36.4.1 starpu_sched_ctx_get_sched_policy_callback

```
void(*)(unsigned) starpu_sched_ctx_get_sched_policy_callback(unsigned sched_ctx_id) (
            unsigned sched_ctx_id )
```

Return the function associated with the scheduler context `sched_ctx_id` which was given through the field starpu_conf::sched_policy_callback

57.37 Scheduling Policy

TODO. While StarPU comes with a variety of scheduling policies (see Task Scheduling Policies), it may sometimes be desirable to implement custom policies to address specific problems. The API described below allows users to write their own scheduling policy.

Data Structures

- struct starpu_sched_policy

Macros

- #define STARPU_NMAX_SCHED_CTXS
- #define STARPU_MAXIMPLEMENTATIONS

Typedefs

- typedef void(∗ **starpu_notify_ready_soon_func**) (void ∗data, struct starpu_task ∗task, double delay)

Functions

- struct starpu_sched_policy ∗∗ starpu_sched_get_predefined_policies (void)
- struct starpu_sched_policy ∗ starpu_get_sched_lib_policy (const char ∗name)
- struct starpu_sched_policy ∗∗ starpu_get_sched_lib_policies (void)
- struct starpu_sched_policy ∗ starpu_sched_get_sched_policy_in_ctx (unsigned sched_ctx_id)
- struct starpu_sched_policy ∗ starpu_sched_get_sched_policy (void)
- void starpu_worker_get_sched_condition (int workerid, starpu_pthread_mutex_t ∗∗sched_mutex, starpu_pthread_cond_t ∗∗sched_cond)
- unsigned long starpu_task_get_job_id (struct starpu_task ∗task)
- int starpu_sched_get_min_priority (void)
- int starpu_sched_get_max_priority (void)
- int starpu_sched_set_min_priority (int min_prio)
- int starpu_sched_set_max_priority (int max_prio)
- int starpu_worker_can_execute_task (unsigned workerid, struct starpu_task ∗task, unsigned nimpl)
- int starpu_worker_can_execute_task_impl (unsigned workerid, struct starpu_task ∗task, unsigned ∗impl_mask)
- int starpu_worker_can_execute_task_first_impl (unsigned workerid, struct starpu_task ∗task, unsigned ∗nimpl)
- int starpu_push_local_task (int workerid, struct starpu_task ∗task, int back)
- int starpu_push_task_end (struct starpu_task ∗task)
- int starpu_get_prefetch_flag (void)
- int starpu_prefetch_task_input_on_node_prio (struct starpu_task ∗task, unsigned node, int prio)
- int starpu_prefetch_task_input_on_node (struct starpu_task ∗task, unsigned node)
- int starpu_idle_prefetch_task_input_on_node_prio (struct starpu_task ∗task, unsigned node, int prio)
- int starpu_idle_prefetch_task_input_on_node (struct starpu_task ∗task, unsigned node)
- int starpu_prefetch_task_input_for_prio (struct starpu_task ∗task, unsigned worker, int prio)
- int starpu_prefetch_task_input_for (struct starpu_task ∗task, unsigned worker)
- int starpu_idle_prefetch_task_input_for_prio (struct starpu_task ∗task, unsigned worker, int prio)
- int starpu_idle_prefetch_task_input_for (struct starpu_task ∗task, unsigned worker)
- uint32_t starpu_task_footprint (struct starpu_perfmodel ∗model, struct starpu_task ∗task, struct starpu_perfmodel_arch ∗arch, unsigned nimpl)
- uint32_t starpu_task_data_footprint (struct starpu_task ∗task)
- double starpu_task_expected_length (struct starpu_task ∗task, struct starpu_perfmodel_arch ∗arch, unsigned nimpl)
- double starpu_task_worker_expected_length (struct starpu_task ∗task, unsigned workerid, unsigned sched_ctx_id, unsigned nimpl)

- double starpu_task_expected_length_average (struct starpu_task ∗task, unsigned sched_ctx_id)
- double starpu_worker_get_relative_speedup (struct starpu_perfmodel_arch ∗perf_arch)
- double starpu_task_expected_data_transfer_time (unsigned memory_node, struct starpu_task ∗task)
- double starpu_task_expected_data_transfer_time_for (struct starpu_task ∗task, unsigned worker)
- double starpu_data_expected_transfer_time (starpu_data_handle_t handle, unsigned memory_node, enum starpu_data_access_mode mode)
- double starpu_task_expected_energy (struct starpu_task ∗task, struct starpu_perfmodel_arch ∗arch, unsigned nimpl)
- double starpu_task_worker_expected_energy (struct starpu_task ∗task, unsigned workerid, unsigned sched_ctx_id, unsigned nimpl)
- double starpu_task_expected_energy_average (struct starpu_task ∗task, unsigned sched_ctx_id)
- double starpu_task_expected_conversion_time (struct starpu_task ∗task, struct starpu_perfmodel_arch ∗arch, unsigned nimpl)
- void starpu_task_notify_ready_soon_register (starpu_notify_ready_soon_func f, void ∗data)
- void starpu_sched_ctx_worker_shares_tasks_lists (int workerid, int sched_ctx_id)
- void starpu_sched_task_break (struct starpu_task ∗task)

Worker operations

- int starpu_wake_worker_relax (int workerid)
- int starpu_wake_worker_no_relax (int workerid)
- int starpu_wake_worker_locked (int workerid)
- int starpu_wake_worker_relax_light (int workerid)

57.37.1 Detailed Description

TODO. While StarPU comes with a variety of scheduling policies (see Task Scheduling Policies), it may sometimes be desirable to implement custom policies to address specific problems. The API described below allows users to write their own scheduling policy.

57.37.2 Data Structure Documentation

57.37.2.1 struct starpu_sched_policy

Contain all the methods that implement a scheduling policy. An application may specify which scheduling strategy in the field starpu_conf::sched_policy passed to the function starpu_init().
For each task going through the scheduler, the following methods get called in the given order:

- starpu_sched_policy::submit_hook when the task is submitted
- starpu_sched_policy::push_task when the task becomes ready. The scheduler is here **given** the task
- starpu_sched_policy::pop_task when the worker is idle. The scheduler here **gives** back the task to the core. It must not access this task any more
- starpu_sched_policy::pre_exec_hook right before the worker actually starts the task computation (after transferring any missing data).
- starpu_sched_policy::post_exec_hook right after the worker actually completes the task computation.

For each task not going through the scheduler (because starpu_task::execute_on_a_specific_worker was set), these get called:

- starpu_sched_policy::submit_hook when the task is submitted
- starpu_sched_policy::push_task_notify when the task becomes ready. This is just a notification, the scheduler does not have to do anything about the task.
- starpu_sched_policy::pre_exec_hook right before the worker actually starts the task computation (after transferring any missing data).
- starpu_sched_policy::post_exec_hook right after the worker actually completes the task computation.

Data Fields

- void(∗ init_sched)(unsigned sched_ctx_id)
- void(∗ deinit_sched)(unsigned sched_ctx_id)
- int(∗ push_task)(struct starpu_task ∗)
- double(∗ **simulate_push_task**)(struct starpu_task ∗)
- void(∗ push_task_notify)(struct starpu_task ∗, int workerid, int perf_workerid, unsigned sched_ctx_id)
- struct starpu_task ∗(∗ pop_task)(unsigned sched_ctx_id)
- void(∗ submit_hook)(struct starpu_task ∗task)
- void(∗ pre_exec_hook)(struct starpu_task ∗, unsigned sched_ctx_id)
- void(∗ post_exec_hook)(struct starpu_task ∗, unsigned sched_ctx_id)
- void(∗ do_schedule)(unsigned sched_ctx_id)
- void(∗ add_workers)(unsigned sched_ctx_id, int ∗workerids, unsigned nworkers)
- void(∗ remove_workers)(unsigned sched_ctx_id, int ∗workerids, unsigned nworkers)
- int prefetches
- const char ∗ policy_name
- const char ∗ policy_description
- enum starpu_worker_collection_type **worker_type**

57.37.2.1.1 Field Documentation

57.37.2.1.1.1 init_sched `void(∗ starpu_sched_policy::init_sched) (unsigned sched_ctx_id)`

Initialize the scheduling policy, called before any other method.

57.37.2.1.1.2 deinit_sched `void(∗ starpu_sched_policy::deinit_sched) (unsigned sched_ctx_id)`

Cleanup the scheduling policy

57.37.2.1.1.3 push_task `int(∗ starpu_sched_policy::push_task) (struct starpu_task ∗)`

Insert a task into the scheduler, called when the task becomes ready for execution. This must call starpu_push_task_end() once it has effectively pushed the task to a queue (to note the time when this was done in the task), but before releasing mutexes (so that the task hasn't been already taken by a worker).

57.37.2.1.1.4 push_task_notify `void(∗ starpu_sched_policy::push_task_notify) (struct starpu_task ∗, int workerid, int perf_workerid, unsigned sched_ctx_id)`

Notify the scheduler that a task was pushed on a given worker. This method is called when a task that was explicitly assigned to a worker becomes ready and is about to be executed by the worker. This method therefore permits to keep the state of the scheduler coherent even when StarPU bypasses the scheduling strategy.
Note: to get an estimation of the task duration, `perf_workerid` needs to be used rather than `workerid`, for the case of parallel tasks.

57.37.2.1.1.5 pop_task `struct starpu_task ∗(∗ starpu_sched_policy::pop_task) (unsigned sched_↩ ctx_id)`

Get a task from the scheduler. If this method returns NULL, the worker will start sleeping. If later on some task are pushed for this worker, starpu_wake_worker() must be called to wake the worker so it can call the pop_task() method again. The mutex associated to the worker is already taken when this method is called. This method may release it (e.g. for scalability reasons when doing work stealing), but it must acquire it again before taking the decision whether to return a task or NULL, so the atomicity of deciding to return NULL and making the worker actually sleep is preserved. Otherwise in simgrid or blocking driver mode the worker might start sleeping while a task has just been pushed for it. If this method is defined as `NULL`, the worker will only execute tasks from its local queue. In this case, the push_task method should use the starpu_push_local_task method to assign tasks to the different workers.

57.37.2.1.1.6 submit_hook `void(∗ starpu_sched_policy::submit_hook) (struct starpu_task ∗task)`

Optional field. This method is called when a task is submitted.

57.37.2.1.1.7 pre_exec_hook `void(* starpu_sched_policy::pre_exec_hook) (struct starpu_task *, unsigned sched_ctx_id)`

Optional field. This method is called every time a task is starting.

57.37.2.1.1.8 post_exec_hook `void(* starpu_sched_policy::post_exec_hook) (struct starpu_task *, unsigned sched_ctx_id)`

Optional field. This method is called every time a task has been executed.

57.37.2.1.1.9 do_schedule `void(* starpu_sched_policy::do_schedule) (unsigned sched_ctx_id)`

Optional field. This method is called when it is a good time to start scheduling tasks. This is notably called when the application calls starpu_task_wait_for_all() or starpu_do_schedule() explicitly.

57.37.2.1.1.10 add_workers `void(* starpu_sched_policy::add_workers) (unsigned sched_ctx_id, int *workerids, unsigned nworkers)`

Initialize scheduling structures corresponding to each worker used by the policy.

57.37.2.1.1.11 remove_workers `void(* starpu_sched_policy::remove_workers) (unsigned sched_↩ ctx_id, int *workerids, unsigned nworkers)`

Deinitialize scheduling structures corresponding to each worker used by the policy.

57.37.2.1.1.12 prefetches `int starpu_sched_policy::prefetches`

Whether this scheduling policy does data prefetching, and thus the core should not try to do it opportunistically.

57.37.2.1.1.13 policy_name `const char* starpu_sched_policy::policy_name`

Optional field. Name of the policy.

57.37.2.1.1.14 policy_description `const char* starpu_sched_policy::policy_description`

Optional field. Human readable description of the policy.

57.37.3 Macro Definition Documentation

57.37.3.1 STARPU_NMAX_SCHED_CTXS

`#define STARPU_NMAX_SCHED_CTXS`

Define the maximum number of scheduling contexts managed by StarPU. The default value can be modified at configure by using the option --enable-max-sched-ctxs.

57.37.3.2 STARPU_MAXIMPLEMENTATIONS

`#define STARPU_MAXIMPLEMENTATIONS`

Define the maximum number of implementations per architecture. The default value can be modified at configure by using the option --enable-maximplementations.

57.37.4 Function Documentation

57.37.4.1 starpu_sched_get_predefined_policies()

```
struct starpu_sched_policy ** starpu_sched_get_predefined_policies (
            void  )
```

Return an `NULL`-terminated array of all the predefined scheduling policies. See Task Scheduling Policies for more details.

57.37.4.2 starpu_get_sched_lib_policy()

```
struct starpu_sched_policy * starpu_get_sched_lib_policy (
            const char * name )
```

Allow an external library to return a scheduling policy to be loaded dynamically. See Using a New Scheduling Policy for more details.

57.37.4.3 starpu_get_sched_lib_policies()

```
struct starpu_sched_policy ** starpu_get_sched_lib_policies (
            void  )
```

Allow an external library to return a list of scheduling policies to be loaded dynamically. See Using a New Scheduling Policy for more details.

57.37.4.4 starpu_sched_get_sched_policy_in_ctx()

```
struct starpu_sched_policy * starpu_sched_get_sched_policy_in_ctx (
            unsigned sched_ctx_id )
```

Return the scheduler policy of the default context. See Task Scheduling Policies for more details.

57.37.4.5 starpu_sched_get_sched_policy()

```
struct starpu_sched_policy * starpu_sched_get_sched_policy (
            void  )
```

Return the scheduler policy of the given context. See Task Scheduling Policies for more details.

57.37.4.6 starpu_worker_get_sched_condition()

```
void starpu_worker_get_sched_condition (
            int workerid,
            starpu_pthread_mutex_t ** sched_mutex,
            starpu_pthread_cond_t ** sched_cond )
```

When there is no available task for a worker, StarPU blocks this worker on a condition variable. This function specifies which condition variable (and the associated mutex) should be used to block (and to wake up) a worker. Note that multiple workers may use the same condition variable. For instance, in the case of a scheduling strategy with a single task queue, the same condition variable would be used to block and wake up all workers.

57.37.4.7 starpu_task_get_job_id()

```
unsigned long starpu_task_get_job_id (
            struct starpu_task * task )
```

Return the job identifier associated with the task. See Getting Scheduling Task Details for more details.

57.37.4.8 starpu_sched_get_min_priority()

```
int starpu_sched_get_min_priority (
            void  )
```

TODO: check if this is correct Return the current minimum priority level supported by the scheduling policy. See Defining A New Basic Scheduling Policy for more details.

57.37.4.9 starpu_sched_get_max_priority()

```
int starpu_sched_get_max_priority (
            void  )
```

TODO: check if this is correct Return the current maximum priority level supported by the scheduling policy. See Defining A New Basic Scheduling Policy for more details.

57.37.4.10 starpu_sched_set_min_priority()

```
int starpu_sched_set_min_priority (
            int min_prio )
```

TODO: check if this is correct Define the minimum task priority level supported by the scheduling policy. The default minimum priority level is the same as the default priority level which is 0 by convention. The application may access that value by calling the function starpu_sched_get_min_priority(). This function should only be called from the initialization method of the scheduling policy, and should not be used directly from the application. See Defining A New Basic Scheduling Policy for more details.

57.37.4.11 starpu_sched_set_max_priority()

```
int starpu_sched_set_max_priority (
            int max_prio )
```

TODO: check if this is correct Define the maximum priority level supported by the scheduling policy. The default maximum priority level is 1. The application may access that value by calling the function starpu_sched_get_max_priority(). This function should only be called from the initialization method of the scheduling policy, and should not be used directly from the application. See Defining A New Basic Scheduling Policy for more details.

57.37.4.12 starpu_worker_can_execute_task()

```
int starpu_worker_can_execute_task (
            unsigned workerid,
            struct starpu_task * task,
            unsigned nimpl )
```

Check if the worker specified by workerid can execute the codelet. Schedulers need to call it before assigning a task to a worker, otherwise the task may fail to execute. See Defining A New Basic Scheduling Policy for more details.

57.37.4.13 starpu_worker_can_execute_task_impl()

```
int starpu_worker_can_execute_task_impl (
            unsigned workerid,
            struct starpu_task * task,
            unsigned * impl_mask )
```

Check if the worker specified by workerid can execute the codelet and return which implementation numbers can be used. Schedulers need to call it before assigning a task to a worker, otherwise the task may fail to execute. This should be preferred rather than calling starpu_worker_can_execute_task() for each and every implementation. It can also be used with `impl_mask == NULL` to check for at least one implementation without determining which. See Defining A New Basic Scheduling Policy for more details.

57.37.4.14 starpu_worker_can_execute_task_first_impl()

```
int starpu_worker_can_execute_task_first_impl (
            unsigned workerid,
            struct starpu_task * task,
            unsigned * nimpl )
```

Check if the worker specified by workerid can execute the codelet and return the first implementation which can be used. Schedulers need to call it before assigning a task to a worker, otherwise the task may fail to execute. This should be preferred rather than calling starpu_worker_can_execute_task() for each and every implementation. It can also be used with `impl_mask == NULL` to check for at least one implementation without determining which. See Defining A New Basic Scheduling Policy for more details.

57.37.4.15 starpu_push_local_task()

```
int starpu_push_local_task (
            int workerid,
            struct starpu_task * task,
            int back )
```

The scheduling policy may put tasks directly into a worker's local queue so that it is not always necessary to create its own queue when the local queue is sufficient. `back` is ignored: the task priority is used to order tasks in this queue. See Defining A New Basic Scheduling Policy for more details.

57.37.4.16 starpu_push_task_end()

```
int starpu_push_task_end (
            struct starpu_task * task )
```

Must be called by a scheduler to notify that the given task has just been pushed. See Defining A New Basic Scheduling Policy for more details.

57.37.4.17 starpu_get_prefetch_flag()

```
int starpu_get_prefetch_flag (
            void  )
```

Whether STARPU_PREFETCH was set. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.37.4.18 starpu_prefetch_task_input_on_node_prio()

```
int starpu_prefetch_task_input_on_node_prio (
            struct starpu_task * task,
            unsigned node,
            int prio )
```

Prefetch data for a given p task on a given p node with a given priority. See Helper functions for defining a scheduling policy (Basic or
for more details.

57.37.4.19 starpu_prefetch_task_input_on_node()

```
int starpu_prefetch_task_input_on_node (
            struct starpu_task * task,
            unsigned node )
```

Prefetch data for a given p task on a given p node. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.37.4.20 starpu_idle_prefetch_task_input_on_node_prio()

```
int starpu_idle_prefetch_task_input_on_node_prio (
            struct starpu_task * task,
            unsigned node,
            int prio )
```

Prefetch data for a given p task on a given p node when the bus is idle with a given priority. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.37.4.21 starpu_idle_prefetch_task_input_on_node()

```
int starpu_idle_prefetch_task_input_on_node (
            struct starpu_task * task,
            unsigned node )
```

Prefetch data for a given p task on a given p node when the bus is idle. See Helper functions for defining a scheduling policy (Basic or
for more details.

57.37.4.22 starpu_prefetch_task_input_for_prio()

```
int starpu_prefetch_task_input_for_prio (
            struct starpu_task * task,
            unsigned worker,
            int prio )
```

Prefetch data for a given p task on a given p worker with a given priority. See Helper functions for defining a scheduling policy (Basic or
for more details.

57.37.4.23 starpu_prefetch_task_input_for()

```
int starpu_prefetch_task_input_for (
            struct starpu_task * task,
            unsigned worker )
```

Prefetch data for a given p task on a given p worker. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.37.4.24 starpu_idle_prefetch_task_input_for_prio()

```
int starpu_idle_prefetch_task_input_for_prio (
            struct starpu_task * task,
            unsigned worker,
            int prio )
```

Prefetch data for a given p task on a given p worker when the bus is idle with a given priority. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.37.4.25 starpu_idle_prefetch_task_input_for()

```
int starpu_idle_prefetch_task_input_for (
            struct starpu_task * task,
            unsigned worker )
```

Prefetch data for a given p task on a given p worker when the bus is idle. See Helper functions for defining a scheduling policy (Basic o
for more details.

57.37.4.26 starpu_task_footprint()

```
uint32_t starpu_task_footprint (
            struct starpu_perfmodel * model,
            struct starpu_task * task,
            struct starpu_perfmodel_arch * arch,
            unsigned nimpl )
```

Return the footprint for a given task, taking into account user-provided perfmodel footprint or size_base functions. See Performance Model Example for more details.

57.37.4.27 starpu_task_data_footprint()

```
uint32_t starpu_task_data_footprint (
            struct starpu_task * task )
```

Return the raw footprint for the data of a given task (without taking into account user-provided functions). See Performance Model Example for more details.

57.37.4.28 starpu_task_expected_length()

```
double starpu_task_expected_length (
            struct starpu_task * task,
            struct starpu_perfmodel_arch * arch,
            unsigned nimpl )
```

Return expected task duration in micro-seconds on a given architecture `arch` using given implementation `nimpl`. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.37.4.29 starpu_task_worker_expected_length()

```
double starpu_task_worker_expected_length (
            struct starpu_task * task,
            unsigned workerid,
            unsigned sched_ctx_id,
            unsigned nimpl )
```

Same as starpu_task_expected_length() but for a precise worker. See Helper functions for defining a scheduling policy (Basic or modu
for more details.

57.37.4.30 starpu_task_expected_length_average()

```
double starpu_task_expected_length_average (
            struct starpu_task * task,
            unsigned sched_ctx_id )
```

Return expected task duration in micro-seconds, averaged over the different workers driven by the scheduler `sched_ctx_id` Note: this is not just the average of the durations using the number of processing units as coefficients, but their efficiency at processing the task, thus the harmonic average of the durations. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.37.4.31 starpu_worker_get_relative_speedup()

```
double starpu_worker_get_relative_speedup (
            struct starpu_perfmodel_arch * perf_arch )
```

Return an estimated speedup factor relative to CPU speed. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.37.4.32 starpu_task_expected_data_transfer_time()

```
double starpu_task_expected_data_transfer_time (
            unsigned memory_node,
            struct starpu_task * task )
```

Return expected data transfer time in micro-seconds for the given `memory_node`. Prefer using starpu_task_expected_data_transfer_
which is more precise. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.37.4.33 starpu_task_expected_data_transfer_time_for()

```
double starpu_task_expected_data_transfer_time_for (
            struct starpu_task * task,
            unsigned worker )
```

Return expected data transfer time in micro-seconds for the given `worker`. See Helper functions for defining a scheduling policy (Bas
for more details.

57.37.4.34 starpu_data_expected_transfer_time()

```
double starpu_data_expected_transfer_time (
            starpu_data_handle_t handle,
            unsigned memory_node,
            enum starpu_data_access_mode mode )
```

Predict the transfer time (in micro-seconds) to move `handle` to a memory node. See Helper functions for defining a scheduling policy
for more details.

57.37.4.35 starpu_task_expected_energy()

```
double starpu_task_expected_energy (
            struct starpu_task * task,
            struct starpu_perfmodel_arch * arch,
            unsigned nimpl )
```

Return expected energy use in J. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.37.4.36 starpu_task_worker_expected_energy()

```
double starpu_task_worker_expected_energy (
            struct starpu_task * task,
            unsigned workerid,
            unsigned sched_ctx_id,
            unsigned nimpl )
```

Same as starpu_task_expected_energy but for a precise worker. See Helper functions for defining a scheduling policy (Basic or modu
for more details.

57.37.4.37 starpu_task_expected_energy_average()

```
double starpu_task_expected_energy_average (
            struct starpu_task * task,
            unsigned sched_ctx_id )
```

Return expected task energy use in J, averaged over the different workers driven by the scheduler `sched_ctx_id` Note: this is not just the average of the energy uses using the number of processing units as coefficients, but their efficiency at processing the task, thus the harmonic average of the energy uses. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.37.4.38 starpu_task_expected_conversion_time()

```
double starpu_task_expected_conversion_time (
            struct starpu_task * task,
            struct starpu_perfmodel_arch * arch,
            unsigned nimpl )
```

Return expected conversion time in ms (multiformat interface only). See Helper functions for defining a scheduling policy (Basic or mo
for more details.

57.37.4.39 starpu_task_notify_ready_soon_register()

```
void starpu_task_notify_ready_soon_register (
            starpu_notify_ready_soon_func f,
            void * data )
```

Register a callback to be called when it is determined when a task will be ready an estimated amount of time from now, because its last dependency has just started and we know how long it will take. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.37.4.40 starpu_sched_ctx_worker_shares_tasks_lists()

```
void starpu_sched_ctx_worker_shares_tasks_lists (
            int workerid,
            int sched_ctx_id )
```

The scheduling policies indicates if the worker may pop tasks from the list of other workers or if there is a central list with task for all the workers. See Defining A New Basic Scheduling Policy for more details.

57.37.4.41 starpu_sched_task_break()

```
void starpu_sched_task_break (
            struct starpu_task * task )
```

The scheduling policy should call this when it makes a scheduling decision for a task. This will possibly stop execution at this point, and then the programmer can inspect local variables etc. to determine why this scheduling decision was done.
See STARPU_TASK_BREAK_ON_SCHED See Defining A New Basic Scheduling Policy for more details.

57.37.4.42 starpu_wake_worker_relax()

```
int starpu_wake_worker_relax (
            int workerid )
```

Wake up `workerid` while temporarily entering the current worker relax state if needed during the waiting process. Return 1 if `workerid` has been woken up or its state_keep_awake flag has been set to `1`, and `0` otherwise (if `workerid` was not in the STATE_SLEEPING or in the STATE_SCHEDULING). See Defining A New Basic Scheduling Policy for more details.

57.37.4.43 starpu_wake_worker_no_relax()

```
int starpu_wake_worker_no_relax (
            int workerid )
```

Must be called to wake up a worker that is sleeping on the cond. Return 0 whenever the worker is not in a sleeping state or has the state_keep_awake flag on. See Defining A New Basic Scheduling Policy for more details.

57.37.4.44 starpu_wake_worker_locked()

```
int starpu_wake_worker_locked (
            int workerid )
```

Version of starpu_wake_worker_no_relax() which assumes that the sched mutex is locked See Defining A New Basic Scheduling Polic for more details.

57.37.4.45 starpu_wake_worker_relax_light()

```
int starpu_wake_worker_relax_light (
            int workerid )
```

Light version of starpu_wake_worker_relax() which, when possible, speculatively set keep_awake on the target worker without waiting for the worker to enter the relax state. See Defining A New Basic Scheduling Policy for more details.

57.38 Scheduling Context Hypervisor - Linear Programming

Functions

- double sc_hypervisor_lp_get_nworkers_per_ctx (int nsched_ctxs, int ntypes_of_workers, double res[nsched_ctxs][ntypes_of_workers], int total_nw[ntypes_of_workers], struct types_of_workers ∗tw, unsigned ∗in_sched_ctxs)
- double sc_hypervisor_lp_get_tmax (int nw, int ∗workers)
- void sc_hypervisor_lp_round_double_to_int (int ns, int nw, double res[ns][nw], int res_rounded[ns][nw])
- void sc_hypervisor_lp_redistribute_resources_in_ctxs (int ns, int nw, int res_rounded[ns][nw], double res[ns][nw], unsigned ∗sched_ctxs, struct types_of_workers ∗tw)
- void sc_hypervisor_lp_distribute_resources_in_ctxs (unsigned ∗sched_ctxs, int ns, int nw, int res_rounded[ns][nw], double res[ns][nw], int ∗workers, int nworkers, struct types_of_workers ∗tw)
- void sc_hypervisor_lp_distribute_floating_no_resources_in_ctxs (unsigned ∗sched_ctxs, int ns, int nw, double res[ns][nw], int ∗workers, int nworkers, struct types_of_workers ∗tw)
- void sc_hypervisor_lp_place_resources_in_ctx (int ns, int nw, double w_in_s[ns][nw], unsigned ∗sched_ctxs, int ∗workers, unsigned do_size, struct types_of_workers ∗tw)
- void sc_hypervisor_lp_share_remaining_resources (int ns, unsigned ∗sched_ctxs, int nworkers, int ∗workers)
- double sc_hypervisor_lp_find_tmax (double t1, double t2)
- unsigned sc_hypervisor_lp_execute_dichotomy (int ns, int nw, double w_in_s[ns][nw], unsigned solve_lp_integer, void ∗specific_data, double tmin, double tmax, double smallest_tmax, double(∗lp_estimated_distrib_func)(int lns, int lnw, double ldraft_w_in_s[ns][nw], unsigned lis_integer, double ltmax, void ∗lspecifc_data))
- double sc_hypervisor_lp_simulate_distrib_flops (int nsched_ctxs, int ntypes_of_workers, double speed[nsched_ctxs][ntypes_of_workers], double flops[nsched_ctxs], double res[nsched_ctxs][ntypes_of_workers], int total_nw[ntypes_of_workers], unsigned sched_ctxs[nsched_ctxs], double vmax)
- double sc_hypervisor_lp_simulate_distrib_tasks (int ns, int nw, int nt, double w_in_s[ns][nw], double tasks[nw][nt], double times[nw][nt], unsigned is_integer, double tmax, unsigned ∗in_sched_ctxs, struct sc_hypervisor_policy_task_pool ∗tmp_task_pools)
- double sc_hypervisor_lp_simulate_distrib_flops_on_sample (int ns, int nw, double final_w_in_s[ns][nw], unsigned is_integer, double tmax, double ∗∗speed, double flops[ns], double ∗∗final_flops_on_w)

57.38.1 Detailed Description

57.38.2 Function Documentation

57.38.2.1 sc_hypervisor_lp_get_nworkers_per_ctx()

```
double sc_hypervisor_lp_get_nworkers_per_ctx (
            int nsched_ctxs,
            int ntypes_of_workers,
            double res[nsched_ctxs][ntypes_of_workers],
            int total_nw[ntypes_of_workers],
            struct types_of_workers * tw,
            unsigned * in_sched_ctxs )
```

return tmax, and compute in table res the nr of workers needed by each context st the system ends up in the smallest tma

57.38.2.2 sc_hypervisor_lp_get_tmax()

```
double sc_hypervisor_lp_get_tmax (
            int nw,
            int * workers )
```

return tmax of the system

57.38.2.3 sc_hypervisor_lp_round_double_to_int()

```
void sc_hypervisor_lp_round_double_to_int (
            int ns,
            int nw,
            double res[ns][nw],
            int res_rounded[ns][nw] )
```

the linear programme determines a rational number of resources for each ctx, we round them depending on the type of resource

57.38.2.4 sc_hypervisor_lp_redistribute_resources_in_ctxs()

```
void sc_hypervisor_lp_redistribute_resources_in_ctxs (
            int ns,
            int nw,
            int res_rounded[ns][nw],
            double res[ns][nw],
            unsigned * sched_ctxs,
            struct types_of_workers * tw )
```

redistribute the resource in contexts by assigning the first x available resources to each one

57.38.2.5 sc_hypervisor_lp_distribute_resources_in_ctxs()

```
void sc_hypervisor_lp_distribute_resources_in_ctxs (
            unsigned * sched_ctxs,
            int ns,
            int nw,
            int res_rounded[ns][nw],
            double res[ns][nw],
            int * workers,
            int nworkers,
            struct types_of_workers * tw )
```

make the first distribution of resource in contexts by assigning the first x available resources to each one

57.38.2.6 sc_hypervisor_lp_distribute_floating_no_resources_in_ctxs()

```
void sc_hypervisor_lp_distribute_floating_no_resources_in_ctxs (
            unsigned * sched_ctxs,
            int ns,
            int nw,
            double res[ns][nw],
            int * workers,
            int nworkers,
            struct types_of_workers * tw )
```

make the first distribution of resource in contexts by assigning the first x available resources to each one, share not integer no of workers

57.38.2.7 sc_hypervisor_lp_place_resources_in_ctx()

```
void sc_hypervisor_lp_place_resources_in_ctx (
            int ns,
            int nw,
            double w_in_s[ns][nw],
            unsigned * sched_ctxs,
            int * workers,
            unsigned do_size,
            struct types_of_workers * tw )
```

place resources in contexts depending on whether they already have workers or not

57.38.2.8 sc_hypervisor_lp_share_remaining_resources()

```
void sc_hypervisor_lp_share_remaining_resources (
            int ns,
            unsigned * sched_ctxs,
            int nworkers,
            int * workers )
```

not used resources are shared between all contexts

57.38.2.9 sc_hypervisor_lp_find_tmax()

```
double sc_hypervisor_lp_find_tmax (
            double t1,
            double t2 )
```

dichotomy btw t1 & t2

57.38.2.10 sc_hypervisor_lp_execute_dichotomy()

```
unsigned sc_hypervisor_lp_execute_dichotomy (
            int ns,
            int nw,
            double w_in_s[ns][nw],
            unsigned solve_lp_integer,
            void * specific_data,
            double tmin,
            double tmax,
            double smallest_tmax,
            double(*)(int lns, int lnw, double ldraft_w_in_s[ns][nw], unsigned lis_integer,
double ltmax, void *lspecifc_data) lp_estimated_distrib_func )
```

execute the lp through dichotomy

57.38.2.11 sc_hypervisor_lp_simulate_distrib_flops()

```
double sc_hypervisor_lp_simulate_distrib_flops (
            int nsched_ctxs,
            int ntypes_of_workers,
            double speed[nsched_ctxs][ntypes_of_workers],
            double flops[nsched_ctxs],
            double res[nsched_ctxs][ntypes_of_workers],
            int total_nw[ntypes_of_workers],
            unsigned sched_ctxs[nsched_ctxs],
            double vmax )
```

linear program that returns 1/tmax, and computes in table res the nr of workers needed by each context st the system ends up in the smallest tmax

57.38.2.12 sc_hypervisor_lp_simulate_distrib_tasks()

```
double sc_hypervisor_lp_simulate_distrib_tasks (
            int ns,
            int nw,
            int nt,
            double w_in_s[ns][nw],
            double tasks[nw][nt],
            double times[nw][nt],
            unsigned is_integer,
            double tmax,
            unsigned * in_sched_ctxs,
            struct sc_hypervisor_policy_task_pool * tmp_task_pools )
```

linear program that simulates a distribution of tasks that minimises the execution time of the tasks in the pool

57.38.2.13 sc_hypervisor_lp_simulate_distrib_flops_on_sample()

```
double sc_hypervisor_lp_simulate_distrib_flops_on_sample (
            int ns,
            int nw,
            double final_w_in_s[ns][nw],
            unsigned is_integer,
            double tmax,
            double ** speed,
            double flops[ns],
            double ** final_flops_on_w )
```

linear program that simulates a distribution of flops over the workers on particular sample of the execution of the application such that the entire sample would finish in a minimum amount of time

57.39 Scheduling Context Hypervisor - Building a new resizing policy

Data Structures

- struct types_of_workers
- struct sc_hypervisor_policy_task_pool
- struct sc_hypervisor_policy
- struct sc_hypervisor_resize_ack

Macros

- #define **HYPERVISOR_REDIM_SAMPLE**
- #define **HYPERVISOR_START_REDIM_SAMPLE**
- #define **SC_NOTHING**
- #define **SC_IDLE**
- #define **SC_SPEED**

Functions

- void sc_hypervisor_policy_add_task_to_pool (struct starpu_codelet ∗cl, unsigned sched_ctx, uint32_t footprint, struct sc_hypervisor_policy_task_pool ∗∗task_pools, size_t data_size)
- void sc_hypervisor_policy_remove_task_from_pool (struct starpu_task ∗task, uint32_t footprint, struct sc_hypervisor_policy_task_pool ∗∗task_pools)
- struct sc_hypervisor_policy_task_pool ∗ sc_hypervisor_policy_clone_task_pool (struct sc_hypervisor_policy_task_pool ∗tp)
- void sc_hypervisor_get_tasks_times (int nw, int nt, double times[nw][nt], int ∗workers, unsigned size_ctxs, struct sc_hypervisor_policy_task_pool ∗task_pools)
- unsigned sc_hypervisor_find_lowest_prio_sched_ctx (unsigned req_sched_ctx, int nworkers_to_move)
- int ∗ sc_hypervisor_get_idlest_workers (unsigned sched_ctx, int ∗nworkers, enum starpu_worker_archtype arch)
- int ∗ sc_hypervisor_get_idlest_workers_in_list (int ∗start, int ∗workers, int nall_workers, int ∗nworkers, enum starpu_worker_archtype arch)
- int sc_hypervisor_get_movable_nworkers (struct sc_hypervisor_policy_config ∗config, unsigned sched_ctx, enum starpu_worker_archtype arch)
- int sc_hypervisor_compute_nworkers_to_move (unsigned req_sched_ctx)
- unsigned sc_hypervisor_policy_resize (unsigned sender_sched_ctx, unsigned receiver_sched_ctx, unsigned force_resize, unsigned now)
- unsigned sc_hypervisor_policy_resize_to_unknown_receiver (unsigned sender_sched_ctx, unsigned now)
- double sc_hypervisor_get_ctx_speed (struct sc_hypervisor_wrapper ∗sc_w)
- double sc_hypervisor_get_slowest_ctx_exec_time (void)
- double sc_hypervisor_get_fastest_ctx_exec_time (void)
- double sc_hypervisor_get_speed_per_worker (struct sc_hypervisor_wrapper ∗sc_w, unsigned worker)
- double sc_hypervisor_get_speed_per_worker_type (struct sc_hypervisor_wrapper ∗sc_w, enum starpu_worker_archtype arch)
- double sc_hypervisor_get_ref_speed_per_worker_type (struct sc_hypervisor_wrapper ∗sc_w, enum starpu_worker_archtype arch)
- double sc_hypervisor_get_avg_speed (enum starpu_worker_archtype arch)
- void sc_hypervisor_check_if_consider_max (struct types_of_workers ∗tw)
- void sc_hypervisor_group_workers_by_type (struct types_of_workers ∗tw, int ∗total_nw)
- enum starpu_worker_archtype sc_hypervisor_get_arch_for_index (unsigned w, struct types_of_workers ∗tw)
- unsigned sc_hypervisor_get_index_for_arch (enum starpu_worker_archtype arch, struct types_of_workers ∗tw)
- unsigned sc_hypervisor_criteria_fulfilled (unsigned sched_ctx, int worker)
- unsigned sc_hypervisor_check_idle (unsigned sched_ctx, int worker)
- unsigned sc_hypervisor_check_speed_gap_btw_ctxs (unsigned ∗sched_ctxs, int nsched_ctxs, int ∗workers, int nworkers)

- unsigned sc_hypervisor_check_speed_gap_btw_ctxs_on_level (int level, int ∗workers_in, int nworkers_in, unsigned father_sched_ctx_id, unsigned ∗∗sched_ctxs, int ∗nsched_ctxs)
- unsigned sc_hypervisor_get_resize_criteria (void)
- struct types_of_workers ∗ sc_hypervisor_get_types_of_workers (int ∗workers, unsigned nworkers)

- struct sc_hypervisor_wrapper ∗ sc_hypervisor_get_wrapper (unsigned sched_ctx)
- unsigned ∗ sc_hypervisor_get_sched_ctxs (void)
- int sc_hypervisor_get_nsched_ctxs (void)
- double sc_hypervisor_get_elapsed_flops_per_sched_ctx (struct sc_hypervisor_wrapper ∗sc_w)
- int sc_hypervisor_get_nworkers_ctx (unsigned sched_ctx, enum starpu_worker_archtype arch)
- double sc_hypervisor_get_total_elapsed_flops_per_sched_ctx (struct sc_hypervisor_wrapper ∗sc_w)
- double sc_hypervisorsc_hypervisor_get_speed_per_worker_type (struct sc_hypervisor_wrapper ∗sc_w, enum starpu_worker_archtype arch)
- double sc_hypervisor_get_speed (struct sc_hypervisor_wrapper ∗sc_w, enum starpu_worker_archtype arch)

- void sc_hypervisor_set_config (unsigned sched_ctx, void ∗config)
- struct sc_hypervisor_policy_config ∗ sc_hypervisor_get_config (unsigned sched_ctx)
- void sc_hypervisor_ctl (unsigned sched_ctx,...)
- #define SC_HYPERVISOR_MAX_IDLE
- #define **SC_HYPERVISOR_MIN_WORKING**
- #define SC_HYPERVISOR_PRIORITY
- #define SC_HYPERVISOR_MIN_WORKERS
- #define SC_HYPERVISOR_MAX_WORKERS
- #define SC_HYPERVISOR_GRANULARITY
- #define SC_HYPERVISOR_FIXED_WORKERS
- #define SC_HYPERVISOR_MIN_TASKS
- #define SC_HYPERVISOR_NEW_WORKERS_MAX_IDLE
- #define SC_HYPERVISOR_TIME_TO_APPLY
- #define SC_HYPERVISOR_NULL
- #define SC_HYPERVISOR_ISPEED_W_SAMPLE
- #define SC_HYPERVISOR_ISPEED_CTX_SAMPLE
- #define **SC_HYPERVISOR_TIME_SAMPLE**
- #define **MAX_IDLE_TIME**
- #define **MIN_WORKING_TIME**

57.39.1 Detailed Description

57.39.2 Data Structure Documentation

57.39.2.1 struct types_of_workers

Data Fields

unsigned	ncpus	
unsigned	ncuda	
unsigned	nw	

57.39.2.2 struct sc_hypervisor_policy_task_pool

Task wrapper linked list

Data Fields

struct starpu_codelet ∗	cl	Which codelet has been executed
uint32_t	footprint	Task footprint key
unsigned	sched_ctx_id	Context the task belongs to

Data Fields

unsigned long	n	Number of tasks of this kind
size_t	data_size	The quantity of data(in bytes) needed by the task to execute
struct sc_hypervisor_policy_task_pool ∗	next	Other task kinds

57.39.2.3 struct sc_hypervisor_policy

Methods to implement a hypervisor resizing policy.

Data Fields

- const char ∗ name
- unsigned custom
- void(∗ size_ctxs)(unsigned ∗sched_ctxs, int nsched_ctxs, int ∗workers, int nworkers)
- void(∗ resize_ctxs)(unsigned ∗sched_ctxs, int nsched_ctxs, int ∗workers, int nworkers)
- void(∗ handle_idle_cycle)(unsigned sched_ctx, int worker)
- void(∗ handle_pushed_task)(unsigned sched_ctx, int worker)
- void(∗ handle_poped_task)(unsigned sched_ctx, int worker, struct starpu_task ∗task, uint32_t footprint)
- void(∗ handle_idle_end)(unsigned sched_ctx, int worker)
- void(∗ handle_post_exec_hook)(unsigned sched_ctx, int task_tag)
- void(∗ handle_submitted_job)(struct starpu_codelet ∗cl, unsigned sched_ctx, uint32_t footprint, size_t data_size)
- void(∗ end_ctx)(unsigned sched_ctx)
- void(∗ start_ctx)(unsigned sched_ctx)
- void(∗ init_worker)(int workerid, unsigned sched_ctx)

57.39.2.3.1 Field Documentation

57.39.2.3.1.1 name `const char* sc_hypervisor_policy::name`
Indicate the name of the policy, if there is not a custom policy, the policy corresponding to this name will be used by the hypervisor

57.39.2.3.1.2 custom `unsigned sc_hypervisor_policy::custom`
Indicate whether the policy is custom or not

57.39.2.3.1.3 size_ctxs `void(* sc_hypervisor_policy::size_ctxs) (unsigned *sched_ctxs, int nsched_ctxs, int *workers, int nworkers)`
Distribute workers to contexts even at the beginning of the program

57.39.2.3.1.4 resize_ctxs `void(* sc_hypervisor_policy::resize_ctxs) (unsigned *sched_ctxs, int nsched_ctxs, int *workers, int nworkers)`
Require explicit resizing

57.39.2.3.1.5 handle_idle_cycle `void(* sc_hypervisor_policy::handle_idle_cycle) (unsigned sched_ctx, int worker)`
Called whenever the indicated worker executes another idle cycle in sched_ctx

57.39.2.3.1.6 handle_pushed_task `void(* sc_hypervisor_policy::handle_pushed_task) (unsigned sched_ctx, int worker)`
Called whenever a task is pushed on the worker’s queue corresponding to the context sched_ctx

57.39.2.3.1.7 handle_poped_task `void(* sc_hypervisor_policy::handle_poped_task) (unsigned sched_ctx, int worker, struct starpu_task *task, uint32_t footprint)`

Called whenever a task is poped from the worker's queue corresponding to the context sched_ctx

57.39.2.3.1.8 handle_idle_end `void(* sc_hypervisor_policy::handle_idle_end) (unsigned sched_ctx, int worker)`

Called whenever a task is executed on the indicated worker and context after a long period of idle time

57.39.2.3.1.9 handle_post_exec_hook `void(* sc_hypervisor_policy::handle_post_exec_hook) (unsigned sched_ctx, int task_tag)`

Called whenever a tag task has just been executed. The table of resize requests is provided as well as the tag

57.39.2.3.1.10 handle_submitted_job `void(* sc_hypervisor_policy::handle_submitted_job) (struct starpu_codelet *cl, unsigned sched_ctx, uint32_t footprint, size_t data_size)`

the hypervisor takes a decision when a job was submitted in this ctx

57.39.2.3.1.11 end_ctx `void(* sc_hypervisor_policy::end_ctx) (unsigned sched_ctx)`

the hypervisor takes a decision when a certain ctx was deleted

57.39.2.3.1.12 start_ctx `void(* sc_hypervisor_policy::start_ctx) (unsigned sched_ctx)`

the hypervisor takes a decision when a certain ctx was registered

57.39.2.3.1.13 init_worker `void(* sc_hypervisor_policy::init_worker) (int workerid, unsigned sched_ctx)`

the hypervisor initializes values for the workers

57.39.2.4 struct sc_hypervisor_resize_ack

Structure to check if the workers moved to another context are actually taken into account in that context.

Data Fields

int	receiver_sched_ctx	The context receiving the new workers
int *	moved_workers	List of workers required to be moved
int	nmoved_workers	Number of workers required to be moved
int *	acked_workers	List of workers that actually got in the receiver ctx. If the value corresponding to a worker is 1, this worker got moved in the new context.

57.39.3 Macro Definition Documentation

57.39.3.1 SC_HYPERVISOR_MAX_IDLE

`#define SC_HYPERVISOR_MAX_IDLE`

This macro is used when calling sc_hypervisor_ctl() and must be followed by 3 arguments: an array of int for the workerids to apply the condition, an int to indicate the size of the array, and a double value indicating the maximum idle time allowed for a worker before the resizing process should be triggered

57.39.3.2 SC_HYPERVISOR_PRIORITY

`#define SC_HYPERVISOR_PRIORITY`

This macro is used when calling sc_hypervisor_ctl() and must be followed by 3 arguments: an array of int for the workerids to apply the condition, an int to indicate the size of the array, and an int value indicating the priority of the workers previously mentioned. The workers with the smallest priority are moved the first.

57.39.3.3 SC_HYPERVISOR_MIN_WORKERS

`#define SC_HYPERVISOR_MIN_WORKERS`

This macro is used when calling sc_hypervisor_ctl() and must be followed by 1 argument(int) indicating the minimum number of workers a context should have, underneath this limit the context cannot execute.

57.39.3.4 SC_HYPERVISOR_MAX_WORKERS

`#define SC_HYPERVISOR_MAX_WORKERS`

This macro is used when calling sc_hypervisor_ctl() and must be followed by 1 argument(int) indicating the maximum number of workers a context should have, above this limit the context would not be able to scale

57.39.3.5 SC_HYPERVISOR_GRANULARITY

`#define SC_HYPERVISOR_GRANULARITY`

This macro is used when calling sc_hypervisor_ctl() and must be followed by 1 argument(int) indicating the granularity of the resizing process (the number of workers should be moved from the context once it is resized) This parameter is ignore for the Gflops rate based strategy (see Resizing Strategies), the number of workers that have to be moved is calculated by the strategy.

57.39.3.6 SC_HYPERVISOR_FIXED_WORKERS

`#define SC_HYPERVISOR_FIXED_WORKERS`

This macro is used when calling sc_hypervisor_ctl() and must be followed by 2 arguments: an array of int for the workerids to apply the condition and an int to indicate the size of the array. These workers are not allowed to be moved from the context.

57.39.3.7 SC_HYPERVISOR_MIN_TASKS

`#define SC_HYPERVISOR_MIN_TASKS`

This macro is used when calling sc_hypervisor_ctl() and must be followed by 1 argument (int) that indicated the minimum number of tasks that have to be executed before the context could be resized. This parameter is ignored for the Application Driven strategy (see Resizing Strategies) where the user indicates exactly when the resize should be done.

57.39.3.8 SC_HYPERVISOR_NEW_WORKERS_MAX_IDLE

`#define SC_HYPERVISOR_NEW_WORKERS_MAX_IDLE`

This macro is used when calling sc_hypervisor_ctl() and must be followed by 1 argument, a double value indicating the maximum idle time allowed for workers that have just been moved from other contexts in the current context.

57.39.3.9 SC_HYPERVISOR_TIME_TO_APPLY

`#define SC_HYPERVISOR_TIME_TO_APPLY`

This macro is used when calling sc_hypervisor_ctl() and must be followed by 1 argument (int) indicating the tag an executed task should have such that this configuration should be taken into account.

57.39.3.10 SC_HYPERVISOR_NULL

`#define SC_HYPERVISOR_NULL`

This macro is used when calling sc_hypervisor_ctl() and must be followed by 1 argument

57.39.3.11 SC_HYPERVISOR_ISPEED_W_SAMPLE

`#define SC_HYPERVISOR_ISPEED_W_SAMPLE`

This macro is used when calling sc_hypervisor_ctl() and must be followed by 1 argument, a double, that indicates the number of flops needed to be executed before computing the speed of a worker

57.39.3.12 SC_HYPERVISOR_ISPEED_CTX_SAMPLE

`#define SC_HYPERVISOR_ISPEED_CTX_SAMPLE`

This macro is used when calling sc_hypervisor_ctl() and must be followed by 1 argument, a double, that indicates the number of flops needed to be executed before computing the speed of a context

57.39.4 Function Documentation

57.39.4.1 sc_hypervisor_policy_add_task_to_pool()

```
void sc_hypervisor_policy_add_task_to_pool (
            struct starpu_codelet * cl,
            unsigned sched_ctx,
            uint32_t footprint,
            struct sc_hypervisor_policy_task_pool ** task_pools,
            size_t data_size )
```

add task information to a task wrapper linked list

57.39.4.2 sc_hypervisor_policy_remove_task_from_pool()

```
void sc_hypervisor_policy_remove_task_from_pool (
            struct starpu_task * task,
            uint32_t footprint,
            struct sc_hypervisor_policy_task_pool ** task_pools )
```

remove task information from a task wrapper linked list

57.39.4.3 sc_hypervisor_policy_clone_task_pool()

```
struct sc_hypervisor_policy_task_pool * sc_hypervisor_policy_clone_task_pool (
            struct sc_hypervisor_policy_task_pool * tp )
```

clone a task wrapper linked list

57.39.4.4 sc_hypervisor_get_tasks_times()

```
void sc_hypervisor_get_tasks_times (
            int nw,
            int nt,
            double times[nw][nt],
            int * workers,
            unsigned size_ctxs,
            struct sc_hypervisor_policy_task_pool * task_pools )
```

get the execution time of the submitted tasks out of starpu's calibration files

57.39.4.5 sc_hypervisor_find_lowest_prio_sched_ctx()

```
unsigned sc_hypervisor_find_lowest_prio_sched_ctx (
            unsigned req_sched_ctx,
            int nworkers_to_move )
```

find the context with the lowest priority in order to move some workers

57.39.4.6 sc_hypervisor_get_idlest_workers()

```
int * sc_hypervisor_get_idlest_workers (
            unsigned sched_ctx,
            int * nworkers,
            enum starpu_worker_archtype arch )
```

find the first most idle workers of a context

57.39.4.7 sc_hypervisor_get_idlest_workers_in_list()

```
int * sc_hypervisor_get_idlest_workers_in_list (
             int * start,
             int * workers,
             int nall_workers,
             int * nworkers,
             enum starpu_worker_archtype arch )
```

find the first most idle workers in a list

57.39.4.8 sc_hypervisor_get_movable_nworkers()

```
int sc_hypervisor_get_movable_nworkers (
             struct sc_hypervisor_policy_config * config,
             unsigned sched_ctx,
             enum starpu_worker_archtype arch )
```

find workers that can be moved from a context (if the constraints of min, max, etc allow this)

57.39.4.9 sc_hypervisor_compute_nworkers_to_move()

```
int sc_hypervisor_compute_nworkers_to_move (
             unsigned req_sched_ctx )
```

compute how many workers should be moved from this context

57.39.4.10 sc_hypervisor_policy_resize()

```
unsigned sc_hypervisor_policy_resize (
             unsigned sender_sched_ctx,
             unsigned receiver_sched_ctx,
             unsigned force_resize,
             unsigned now )
```

check the policy's constraints in order to resize

57.39.4.11 sc_hypervisor_policy_resize_to_unknown_receiver()

```
unsigned sc_hypervisor_policy_resize_to_unknown_receiver (
             unsigned sender_sched_ctx,
             unsigned now )
```

check the policy's constraints in order to resize and find a context willing the resources

57.39.4.12 sc_hypervisor_get_ctx_speed()

```
double sc_hypervisor_get_ctx_speed (
             struct sc_hypervisor_wrapper * sc_w )
```

compute the speed of a context

57.39.4.13 sc_hypervisor_get_slowest_ctx_exec_time()

```
double sc_hypervisor_get_slowest_ctx_exec_time (
             void  )
```

get the time of execution of the slowest context

57.39.4.14 sc_hypervisor_get_fastest_ctx_exec_time()

```
double sc_hypervisor_get_fastest_ctx_exec_time (
             void  )
```

get the time of execution of the fastest context

57.39.4.15 sc_hypervisor_get_speed_per_worker()

```
double sc_hypervisor_get_speed_per_worker (
            struct sc_hypervisor_wrapper * sc_w,
            unsigned worker )
```

compute the speed of a workers in a context

57.39.4.16 sc_hypervisor_get_speed_per_worker_type()

```
double sc_hypervisor_get_speed_per_worker_type (
            struct sc_hypervisor_wrapper * sc_w,
            enum starpu_worker_archtype arch )
```

compute the speed of a type of worker in a context

57.39.4.17 sc_hypervisor_get_ref_speed_per_worker_type()

```
double sc_hypervisor_get_ref_speed_per_worker_type (
            struct sc_hypervisor_wrapper * sc_w,
            enum starpu_worker_archtype arch )
```

compute the speed of a type of worker in a context depending on its history

57.39.4.18 sc_hypervisor_get_avg_speed()

```
double sc_hypervisor_get_avg_speed (
            enum starpu_worker_archtype arch )
```

compute the average speed of a type of worker in all ctxs from the beginning of appl

57.39.4.19 sc_hypervisor_check_if_consider_max()

```
void sc_hypervisor_check_if_consider_max (
            struct types_of_workers * tw )
```

verify if we need to consider the max in the lp

57.39.4.20 sc_hypervisor_group_workers_by_type()

```
void sc_hypervisor_group_workers_by_type (
            struct types_of_workers * tw,
            int * total_nw )
```

get the list of workers grouped by type

57.39.4.21 sc_hypervisor_get_arch_for_index()

```
enum starpu_worker_archtype sc_hypervisor_get_arch_for_index (
            unsigned w,
            struct types_of_workers * tw )
```

get what type of worker corresponds to a certain index of types of workers

57.39.4.22 sc_hypervisor_get_index_for_arch()

```
unsigned sc_hypervisor_get_index_for_arch (
            enum starpu_worker_archtype arch,
            struct types_of_workers * tw )
```

get the index of types of workers corresponding to the type of workers indicated

57.39.4.23 sc_hypervisor_criteria_fulfilled()

```
unsigned sc_hypervisor_criteria_fulfilled (
            unsigned sched_ctx,
            int worker )
```

check if we trigger resizing or not

57.39.4.24 sc_hypervisor_check_idle()

```
unsigned sc_hypervisor_check_idle (
             unsigned sched_ctx,
             int worker )
```

check if worker was idle long enough

57.39.4.25 sc_hypervisor_check_speed_gap_btw_ctxs()

```
unsigned sc_hypervisor_check_speed_gap_btw_ctxs (
             unsigned * sched_ctxs,
             int nsched_ctxs,
             int * workers,
             int nworkers )
```

check if there is a speed gap btw ctxs

57.39.4.26 sc_hypervisor_check_speed_gap_btw_ctxs_on_level()

```
unsigned sc_hypervisor_check_speed_gap_btw_ctxs_on_level (
             int level,
             int * workers_in,
             int nworkers_in,
             unsigned father_sched_ctx_id,
             unsigned ** sched_ctxs,
             int * nsched_ctxs )
```

check if there is a speed gap btw ctxs on one level

57.39.4.27 sc_hypervisor_get_resize_criteria()

```
unsigned sc_hypervisor_get_resize_criteria (
             void )
```

check what triggers resizing (idle, speed, etc.

57.39.4.28 sc_hypervisor_get_types_of_workers()

```
struct types_of_workers * sc_hypervisor_get_types_of_workers (
             int * workers,
             unsigned nworkers )
```

load information concerning the type of workers into a types_of_workers struct

57.39.4.29 sc_hypervisor_get_wrapper()

```
struct sc_hypervisor_wrapper * sc_hypervisor_get_wrapper (
             unsigned sched_ctx )
```

Return the wrapper of the given context

57.39.4.30 sc_hypervisor_get_sched_ctxs()

```
unsigned * sc_hypervisor_get_sched_ctxs (
             void )
```

Get the list of registered contexts

57.39.4.31 sc_hypervisor_get_nsched_ctxs()

```
int sc_hypervisor_get_nsched_ctxs (
             void )
```

Get the number of registered contexts

57.39.4.32 sc_hypervisor_get_elapsed_flops_per_sched_ctx()

```
double sc_hypervisor_get_elapsed_flops_per_sched_ctx (
            struct sc_hypervisor_wrapper * sc_w )
```

Get the number of flops executed by a context since last resizing (reset to 0 when a resizing is done)

57.39.4.33 sc_hypervisor_set_config()

```
void sc_hypervisor_set_config (
            unsigned sched_ctx,
            void * config )
```

Specify the configuration for a context

57.39.4.34 sc_hypervisor_get_config()

```
struct sc_hypervisor_policy_config * sc_hypervisor_get_config (
            unsigned sched_ctx )
```

Return the configuration of a context

57.39.4.35 sc_hypervisor_ctl()

```
void sc_hypervisor_ctl (
            unsigned sched_ctx,
             ... )
```

Specify different parameters for the configuration of a context. The list must be zero-terminated

57.39.4.36 sc_hypervisor_get_nworkers_ctx()

```
int sc_hypervisor_get_nworkers_ctx (
            unsigned sched_ctx,
            enum starpu_worker_archtype arch )
```

Get the number of workers of a certain architecture in a context

57.39.4.37 sc_hypervisor_get_total_elapsed_flops_per_sched_ctx()

```
double sc_hypervisor_get_total_elapsed_flops_per_sched_ctx (
            struct sc_hypervisor_wrapper * sc_w )
```

Get the number of flops executed by a context since the beginning

57.39.4.38 sc_hypervisorsc_hypervisor_get_speed_per_worker_type()

```
double sc_hypervisorsc_hypervisor_get_speed_per_worker_type (
            struct sc_hypervisor_wrapper * sc_w,
            enum starpu_worker_archtype arch )
```

Compute an average value of the cpu/cuda speed

57.39.4.39 sc_hypervisor_get_speed()

```
double sc_hypervisor_get_speed (
            struct sc_hypervisor_wrapper * sc_w,
            enum starpu_worker_archtype arch )
```

Compte the actual speed of all workers of a specific type of worker

57.40 Scheduling Context Hypervisor - Regular usage

Functions

- void ∗ sc_hypervisor_init (struct sc_hypervisor_policy ∗policy)
- void sc_hypervisor_shutdown (void)
- void sc_hypervisor_register_ctx (unsigned sched_ctx, double total_flops)
- void sc_hypervisor_unregister_ctx (unsigned sched_ctx)
- void sc_hypervisor_post_resize_request (unsigned sched_ctx, int task_tag)
- void sc_hypervisor_resize_ctxs (unsigned ∗sched_ctxs, int nsched_ctxs, int ∗workers, int nworkers)
- void sc_hypervisor_stop_resize (unsigned sched_ctx)
- void sc_hypervisor_start_resize (unsigned sched_ctx)
- const char ∗ sc_hypervisor_get_policy (void)
- void sc_hypervisor_add_workers_to_sched_ctx (int ∗workers_to_add, unsigned nworkers_to_add, unsigned sched_ctx)
- void sc_hypervisor_remove_workers_from_sched_ctx (int ∗workers_to_remove, unsigned nworkers_to_↩ remove, unsigned sched_ctx, unsigned now)
- void sc_hypervisor_move_workers (unsigned sender_sched_ctx, unsigned receiver_sched_ctx, int ∗workers_to_move, unsigned nworkers_to_move, unsigned now)
- void sc_hypervisor_size_ctxs (unsigned ∗sched_ctxs, int nsched_ctxs, int ∗workers, int nworkers)
- unsigned sc_hypervisor_get_size_req (unsigned ∗∗sched_ctxs, int ∗nsched_ctxs, int ∗∗workers, int ∗nworkers)
- void sc_hypervisor_save_size_req (unsigned ∗sched_ctxs, int nsched_ctxs, int ∗workers, int nworkers)
- void sc_hypervisor_free_size_req (void)
- unsigned sc_hypervisor_can_resize (unsigned sched_ctx)
- void sc_hypervisor_set_type_of_task (struct starpu_codelet ∗cl, unsigned sched_ctx, uint32_t footprint, size_t data_size)
- void sc_hypervisor_update_diff_total_flops (unsigned sched_ctx, double diff_total_flops)
- void sc_hypervisor_update_diff_elapsed_flops (unsigned sched_ctx, double diff_task_flops)
- void sc_hypervisor_update_resize_interval (unsigned ∗sched_ctxs, int nsched_ctxs, int max_nworkers)
- void sc_hypervisor_get_ctxs_on_level (unsigned ∗∗sched_ctxs, int ∗nsched_ctxs, unsigned hierarchy_level, unsigned father_sched_ctx_id)
- unsigned sc_hypervisor_get_nhierarchy_levels (void)
- void sc_hypervisor_get_leaves (unsigned ∗sched_ctxs, int nsched_ctxs, unsigned ∗leaves, int ∗nleaves)
- double sc_hypervisor_get_nready_flops_of_all_sons_of_sched_ctx (unsigned sched_ctx)
- void **sc_hypervisor_print_overhead** (void)
- void **sc_hypervisor_init_worker** (int workerid, unsigned sched_ctx)

Variables

- starpu_pthread_mutex_t act_hypervisor_mutex

57.40.1 Detailed Description

There is a single hypervisor that is in charge of resizing contexts and the resizing strategy is chosen at the initialization of the hypervisor. A single resize can be done at a time.
The Scheduling Context Hypervisor Plugin provides a series of performance counters to StarPU. By incrementing them, StarPU can help the hypervisor in the resizing decision making process.
The function sc_hypervisor_init() initializes the hypervisor to use the strategy provided as parameter and creates the performance counters (see starpu_sched_ctx_performance_counters). These performance counters represent actually some callbacks that will be used by the contexts to notify the information needed by the hypervisor.
Scheduling Contexts that have to be resized by the hypervisor must be first registered to the hypervisor using the function sc_hypervisor_register_ctx()
Note: The Hypervisor is actually a worker that takes this role once certain conditions trigger the resizing process (there is no additional thread assigned to the hypervisor).

57.40.2 Function Documentation

57.40.2.1 sc_hypervisor_init()

```
void * sc_hypervisor_init (
            struct sc_hypervisor_policy * policy )
```

Start the hypervisor with the given policy

57.40.2.2 sc_hypervisor_shutdown()

```
void sc_hypervisor_shutdown (
            void )
```

Shutdown the hypervisor. The hypervisor and all information concerning it is cleaned. There is no synchronization between this function and starpu_shutdown(). Thus, this should be called after starpu_shutdown(), because the performance counters will still need allocated callback functions.

57.40.2.3 sc_hypervisor_register_ctx()

```
void sc_hypervisor_register_ctx (
            unsigned sched_ctx,
            double total_flops )
```

Register the context to the hypervisor, and indicate the number of flops the context will execute (used for Gflops rate based strategy)

57.40.2.4 sc_hypervisor_unregister_ctx()

```
void sc_hypervisor_unregister_ctx (
            unsigned sched_ctx )
```

Unregister a context from the hypervisor, and so exclude the context from the resizing process

57.40.2.5 sc_hypervisor_post_resize_request()

```
void sc_hypervisor_post_resize_request (
            unsigned sched_ctx,
            int task_tag )
```

Require resizing the context `sched_ctx` whenever a task tagged with the id `task_tag` finished executing

57.40.2.6 sc_hypervisor_resize_ctxs()

```
void sc_hypervisor_resize_ctxs (
            unsigned * sched_ctxs,
            int nsched_ctxs,
            int * workers,
            int nworkers )
```

Require reconsidering the distribution of resources over the indicated scheduling contexts, i.e reevaluate the distribution of the resources and eventually resize if needed

57.40.2.7 sc_hypervisor_stop_resize()

```
void sc_hypervisor_stop_resize (
            unsigned sched_ctx )
```

Do not allow the hypervisor to resize a context.

57.40.2.8 sc_hypervisor_start_resize()

```
void sc_hypervisor_start_resize (
            unsigned sched_ctx )
```

Allow the hypervisor to resize a context if necessary.

57.40.2.9 sc_hypervisor_get_policy()

```
const char * sc_hypervisor_get_policy (
            void )
```

Return the name of the resizing policy used by the hypervisor

57.40.2.10 sc_hypervisor_add_workers_to_sched_ctx()

```
void sc_hypervisor_add_workers_to_sched_ctx (
            int * workers_to_add,
            unsigned nworkers_to_add,
            unsigned sched_ctx )
```

Ask the hypervisor to add workers to a sched_ctx

57.40.2.11 sc_hypervisor_remove_workers_from_sched_ctx()

```
void sc_hypervisor_remove_workers_from_sched_ctx (
            int * workers_to_remove,
            unsigned nworkers_to_remove,
            unsigned sched_ctx,
            unsigned now )
```

Ask the hypervisor to remove workers from a sched_ctx

57.40.2.12 sc_hypervisor_move_workers()

```
void sc_hypervisor_move_workers (
            unsigned sender_sched_ctx,
            unsigned receiver_sched_ctx,
            int * workers_to_move,
            unsigned nworkers_to_move,
            unsigned now )
```

Ask the hypervisor to move workers from one context to another

57.40.2.13 sc_hypervisor_size_ctxs()

```
void sc_hypervisor_size_ctxs (
            unsigned * sched_ctxs,
            int nsched_ctxs,
            int * workers,
            int nworkers )
```

Ask the hypervisor to choose a distribution of workers in the required contexts

57.40.2.14 sc_hypervisor_get_size_req()

```
unsigned sc_hypervisor_get_size_req (
            unsigned ** sched_ctxs,
            int * nsched_ctxs,
            int ** workers,
            int * nworkers )
```

Check if there are pending demands of resizing

57.40.2.15 sc_hypervisor_save_size_req()

```
void sc_hypervisor_save_size_req (
            unsigned * sched_ctxs,
            int nsched_ctxs,
            int * workers,
            int nworkers )
```

Save a demand of resizing

57.40.2.16 sc_hypervisor_free_size_req()

```
void sc_hypervisor_free_size_req (
            void )
```

Clear the list of pending demands of resizing

57.40.2.17 sc_hypervisor_can_resize()

```
unsigned sc_hypervisor_can_resize (
            unsigned sched_ctx )
```

Check out if a context can be resized

57.40.2.18 sc_hypervisor_set_type_of_task()

```
void sc_hypervisor_set_type_of_task (
            struct starpu_codelet * cl,
            unsigned sched_ctx,
            uint32_t footprint,
            size_t data_size )
```

Indicate the types of tasks a context will execute in order to better decide the sizing of ctxs

57.40.2.19 sc_hypervisor_update_diff_total_flops()

```
void sc_hypervisor_update_diff_total_flops (
            unsigned sched_ctx,
            double diff_total_flops )
```

Change dynamically the total number of flops of a context, move the deadline of the finishing time of the context

57.40.2.20 sc_hypervisor_update_diff_elapsed_flops()

```
void sc_hypervisor_update_diff_elapsed_flops (
            unsigned sched_ctx,
            double diff_task_flops )
```

Change dynamically the number of the elapsed flops in a context, modify the past in order to better compute the speed

57.40.2.21 sc_hypervisor_update_resize_interval()

```
void sc_hypervisor_update_resize_interval (
            unsigned * sched_ctxs,
            int nsched_ctxs,
            int max_nworkers )
```

Update the min and max workers needed by each context

57.40.2.22 sc_hypervisor_get_ctxs_on_level()

```
void sc_hypervisor_get_ctxs_on_level (
            unsigned ** sched_ctxs,
            int * nsched_ctxs,
            unsigned hierarchy_level,
            unsigned father_sched_ctx_id )
```

Return a list of contexts that are on the same level in the hierarchy of contexts

57.40.2.23 sc_hypervisor_get_nhierarchy_levels()

```
unsigned sc_hypervisor_get_nhierarchy_levels (
            void )
```

Returns the number of levels of ctxs registered to the hyp

57.40.2.24 sc_hypervisor_get_leaves()

```
void sc_hypervisor_get_leaves (
            unsigned * sched_ctxs,
            int nsched_ctxs,
            unsigned * leaves,
            int * nleaves )
```

Return the leaves ctxs from the list of ctxs

57.40.2.25 sc_hypervisor_get_nready_flops_of_all_sons_of_sched_ctx()

```
double sc_hypervisor_get_nready_flops_of_all_sons_of_sched_ctx (
            unsigned sched_ctx )
```

Return the nready flops of all ctxs below in hierarchy of sched_ctx

57.40.3 Variable Documentation

57.40.3.1 act_hypervisor_mutex

```
starpu_pthread_mutex_t act_hypervisor_mutex  [extern]
```

synchronise the hypervisor when several workers try to update its information

57.41 Sink

Functions

- void **starpu_sink_common_worker** (int argc, char ∗∗argv)

57.41.1 Detailed Description

57.42 Standard Memory Library

Macros

- #define STARPU_MALLOC_PINNED
- #define STARPU_MALLOC_COUNT
- #define STARPU_MALLOC_NORECLAIM
- #define STARPU_MEMORY_WAIT
- #define STARPU_MEMORY_OVERFLOW
- #define STARPU_MALLOC_SIMULATION_FOLDED
- #define STARPU_MALLOC_SIMULATION_UNIQUE
- #define starpu_data_malloc_pinned_if_possible
- #define starpu_data_free_pinned_if_possible

Typedefs

- typedef int(∗ **starpu_malloc_hook**) (unsigned dst_node, void ∗∗A, size_t dim, int flags)
- typedef int(∗ **starpu_free_hook**) (unsigned dst_node, void ∗A, size_t dim, int flags)

Functions

- void starpu_malloc_set_align (size_t align)
- int starpu_malloc (void ∗∗A, size_t dim)
- int starpu_free (void ∗A)
- int starpu_malloc_flags (void ∗∗A, size_t dim, int flags)
- int starpu_free_flags (void ∗A, size_t dim, int flags)
- int starpu_free_noflag (void ∗A, size_t dim)
- void starpu_malloc_set_hooks (starpu_malloc_hook malloc_hook, starpu_free_hook free_hook)
- int starpu_memory_pin (void ∗addr, size_t size)
- int starpu_memory_unpin (void ∗addr, size_t size)
- starpu_ssize_t starpu_memory_get_total (unsigned node)
- starpu_ssize_t starpu_memory_get_available (unsigned node)
- size_t starpu_memory_get_used (unsigned node)
- starpu_ssize_t starpu_memory_get_total_all_nodes (void)
- starpu_ssize_t starpu_memory_get_available_all_nodes (void)
- size_t starpu_memory_get_used_all_nodes (void)
- int starpu_memory_allocate (unsigned node, size_t size, int flags)
- void starpu_memory_deallocate (unsigned node, size_t size)
- void starpu_memory_wait_available (unsigned node, size_t size)
- void starpu_sleep (float nb_sec)
- void starpu_usleep (float nb_micro_sec)
- void starpu_energy_use (float joules)
- double starpu_energy_used (void)

57.42.1 Detailed Description

57.42.2 Macro Definition Documentation

57.42.2.1 STARPU_MALLOC_PINNED

`#define STARPU_MALLOC_PINNED`

Value passed to the function starpu_malloc_flags() to indicate the memory allocation should be pinned.

57.42.2.2 STARPU_MALLOC_COUNT

`#define STARPU_MALLOC_COUNT`
Value passed to the function starpu_malloc_flags() to indicate the memory allocation should be in the limit defined by the environment variables STARPU_LIMIT_CUDA_devid_MEM, STARPU_LIMIT_CUDA_MEM, STARPU_LIMIT_OPENCL_devid_MEM, STARPU_LIMIT_OPENCL_MEM, STARPU_LIMIT_HIP_MEM, STARPU_LIMIT_HIP_devid_
and STARPU_LIMIT_CPU_MEM (see Section How to Limit Memory Used By StarPU And Cache Buffer Allocations). If no memory is available, it tries to reclaim memory from StarPU. Memory allocated this way needs to be freed by calling the function starpu_free_flags() with the same flag.

57.42.2.3 STARPU_MALLOC_NORECLAIM

`#define STARPU_MALLOC_NORECLAIM`
Value passed to the function starpu_malloc_flags() along STARPU_MALLOC_COUNT to indicate that while the memory allocation should be kept in the limits defined for STARPU_MALLOC_COUNT, no reclaiming should be performed by starpu_malloc_flags() itself, thus potentially overflowing the memory node a bit. StarPU will reclaim memory after next task termination, according to the STARPU_MINIMUM_AVAILABLE_MEM, STARPU_TARGET_AVAILABLE_MEM, STARPU_MINIMUM_CLEAN_BUFFERS, and STARPU_TARGET_CLEAN_BUFFERS environment variables. If STARPU_MEMORY_WAIT is set, no overflowing will happen, starpu_malloc_flags() will wait for other eviction mechanisms to release enough memory.

57.42.2.4 STARPU_MEMORY_WAIT

`#define STARPU_MEMORY_WAIT`
Value passed to starpu_memory_allocate() to specify that the function should wait for the requested amount of memory to become available, and atomically allocate it.

57.42.2.5 STARPU_MEMORY_OVERFLOW

`#define STARPU_MEMORY_OVERFLOW`
Value passed to starpu_memory_allocate() to specify that the function should allocate the amount of memory, even if that means overflowing the total size of the memory node.

57.42.2.6 STARPU_MALLOC_SIMULATION_FOLDED

`#define STARPU_MALLOC_SIMULATION_FOLDED`
Value passed to the function starpu_malloc_flags() to indicate that when StarPU is using simgrid, the allocation can be "folded", i.e. a memory area is allocated, but its content is actually a replicate of the same memory area, to avoid having to actually allocate that much memory . This thus allows to have a memory area that does not actually consumes memory, to which one can read from and write to normally, but get bogus values.

57.42.2.7 STARPU_MALLOC_SIMULATION_UNIQUE

`#define STARPU_MALLOC_SIMULATION_UNIQUE`
Value passed to the function starpu_malloc_flags() to indicate that when StarPU is using simgrid, the allocation for that size could be unique. Different from only STARPU_MALLOC_SIMULATION_FOLDED, the same address will be given for all mallocs of that particular size.

57.42.2.8 starpu_data_malloc_pinned_if_possible

`#define starpu_data_malloc_pinned_if_possible`

Deprecated Equivalent to starpu_malloc(). This macro is provided to avoid breaking old codes.

57.42.2.9 starpu_data_free_pinned_if_possible

`#define starpu_data_free_pinned_if_possible`

Deprecated Equivalent to starpu_free(). This macro is provided to avoid breaking old codes.

57.42.3 Function Documentation

57.42.3.1 starpu_malloc_set_align()

```
void starpu_malloc_set_align (
            size_t align )
```

Set an alignment constraints for starpu_malloc() allocations. `align` must be a power of two. This is for instance called automatically by the OpenCL driver to specify its own alignment constraints. See Data Management Allocation for more details.

57.42.3.2 starpu_malloc()

```
int starpu_malloc (
            void ** A,
            size_t dim )
```

Allocate data of the given size `dim` in main memory, and return the pointer to the allocated data through `A`. It will also try to pin it in CUDA or OpenCL, so that data transfers from this buffer can be asynchronous, and thus permit data transfer and computation overlapping. The allocated buffer must be freed thanks to the starpu_free_noflag() function. See Data Management Allocation for more details.

57.42.3.3 starpu_free()

```
int starpu_free (
            void * A )
```

Deprecated Free memory which has previously been allocated with starpu_malloc(). This function is deprecated, one should use starpu_free_noflag(). See Data Management Allocation for more details.

57.42.3.4 starpu_malloc_flags()

```
int starpu_malloc_flags (
            void ** A,
            size_t dim,
            int flags )
```

Perform a memory allocation based on the constraints defined by the given flag. See How to Limit Memory Used By StarPU And Cach
for more details.

57.42.3.5 starpu_free_flags()

```
int starpu_free_flags (
            void * A,
            size_t dim,
            int flags )
```

Free memory by specifying its size. The given flags should be consistent with the ones given to starpu_malloc_flags() when allocating the memory. See How to Limit Memory Used By StarPU And Cache Buffer Allocations for more details.

57.42.3.6 starpu_free_noflag()

```
int starpu_free_noflag (
            void * A,
            size_t dim )
```

Free memory by specifying its size. Should be used for memory allocated with starpu_malloc(). See Data Management Allocation for more details.

57.42.3.7 starpu_malloc_set_hooks()

```
void starpu_malloc_set_hooks (
            starpu_malloc_hook malloc_hook,
            starpu_free_hook free_hook )
```

Set allocation functions to be used by StarPU. By default, StarPU will use `malloc()` (or `cudaHostAlloc()` if CUDA GPUs are used) for all its data handle allocations. The application can specify another allocation primitive by calling this. The malloc_hook should pass the allocated pointer through the `A` parameter, and return 0 on success. On allocation failure, it should return -ENOMEM. The `flags` parameter contains STARPU_MALLOC_PINNED if the memory should be pinned by the hook for GPU transfer efficiency. The hook can use starpu_memory_pin() to achieve this. The `dst_node` parameter is the starpu memory node, one can convert it to an hwloc logical id with starpu_memory_nodes_numa_id_to_hwloclogid() or to an OS NUMA number with starpu_memory_nodes_numa_devid_to_id(). See Data Management Allocation for more details.

57.42.3.8 starpu_memory_pin()

```
int starpu_memory_pin (
            void * addr,
            size_t size )
```

Pin the given memory area, so that CPU-GPU transfers can be done asynchronously with DMAs. The memory must be unpinned with starpu_memory_unpin() before being freed. Return 0 on success, -1 on error. See Data Management Allocation for more details.

57.42.3.9 starpu_memory_unpin()

```
int starpu_memory_unpin (
            void * addr,
            size_t size )
```

Unpin the given memory area previously pinned with starpu_memory_pin(). Return 0 on success, -1 on error. See Data Management Allocation for more details.

57.42.3.10 starpu_memory_get_total()

```
starpu_ssize_t starpu_memory_get_total (
            unsigned node )
```

If a memory limit is defined on the given node (see Section How to Limit Memory Used By StarPU And Cache Buffer Allocations), return the amount of total memory on the node. Otherwise return -1. See How to Limit Memory Used By StarPU And Cache Buffer Allo for more details.

57.42.3.11 starpu_memory_get_available()

```
starpu_ssize_t starpu_memory_get_available (
            unsigned node )
```

If a memory limit is defined on the given node (see Section How to Limit Memory Used By StarPU And Cache Buffer Allocations), return the amount of available memory on the node. Otherwise return -1. See How to Limit Memory Used By StarPU And Cache Buffe for more details.

57.42.3.12 starpu_memory_get_used()

```
size_t starpu_memory_get_used (
            unsigned node )
```

Return the amount of used memory on the node. See Data Management Allocation for more details.

57.42.3.13 starpu_memory_get_total_all_nodes()

```
starpu_ssize_t starpu_memory_get_total_all_nodes (
            void )
```

Return the amount of total memory on all memory nodes for whose a memory limit is defined (see Section Data Management Allocation).

57.42.3.14 starpu_memory_get_available_all_nodes()

```
starpu_ssize_t starpu_memory_get_available_all_nodes (
            void )
```

Return the amount of available memory on all memory nodes for whose a memory limit is defined (see Section Data Management Allocation).

57.42.3.15 starpu_memory_get_used_all_nodes()

```
size_t starpu_memory_get_used_all_nodes (
            void )
```

Return the amount of used memory on all memory nodes. See Data Management Allocation for more details.

57.42.3.16 starpu_memory_allocate()

```
int starpu_memory_allocate (
            unsigned node,
            size_t size,
            int flags )
```

If a memory limit is defined on the given node (see Section How to Limit Memory Used By StarPU And Cache Buffer Allocations), try to allocate some of it. This does not actually allocate memory, but only accounts for it. This can be useful when the application allocates data another way, but want StarPU to be aware of the allocation size e.g. for memory reclaiming. By default, return `-ENOMEM` if there is not enough room on the given node. `flags` can be either STARPU_MEMORY_WAIT or STARPU_MEMORY_OVERFLOW to change this. See How to Limit Memory Used By StarPU And Cache Buffer Allocations for more details.

57.42.3.17 starpu_memory_deallocate()

```
void starpu_memory_deallocate (
            unsigned node,
            size_t size )
```

If a memory limit is defined on the given node (see Section How to Limit Memory Used By StarPU And Cache Buffer Allocations), free some of it. This does not actually free memory, but only accounts for it, like starpu_memory_allocate(). The amount does not have to be exactly the same as what was passed to starpu_memory_allocate(), only the eventual amount needs to be the same, i.e. one call to starpu_memory_allocate() can be followed by several calls to starpu_memory_deallocate() to declare the deallocation piece by piece. See How to Limit Memory Used By StarPU And Cache Buffer Allocations for more details.

57.42.3.18 starpu_memory_wait_available()

```
void starpu_memory_wait_available (
            unsigned node,
            size_t size )
```

If a memory limit is defined on the given node (see Section How to Limit Memory Used By StarPU And Cache Buffer Allocations), this will wait for `size` bytes to become available on `node`. Of course, since another thread may be allocating memory concurrently, this does not necessarily mean that this amount will be actually available, just that it was reached. To atomically wait for some amount of memory and reserve it, starpu_memory_allocate() should be used with the STARPU_MEMORY_WAIT flag. See How to Limit Memory Used By StarPU And Cache Buffer Allocations for more details.

57.42.3.19 starpu_sleep()

```
void starpu_sleep (
            float nb_sec )
```

Sleep for the given `nb_sec` seconds. Similar to calling Unix' `sleep` function, except that it takes a float to allow sub-second sleeping, and when StarPU is compiled in SimGrid mode it does not really sleep but just makes SimGrid record that the thread has taken some time to sleep. See Helpers for more details.

57.42.3.20 starpu_usleep()

```
void starpu_usleep (
            float nb_micro_sec )
```

Sleep for the given `nb_micro_sec` micro-seconds. In simgrid mode, this only sleeps within virtual time. See Helpers for more details.

57.42.3.21 starpu_energy_use()

```
void starpu_energy_use (
            float joules )
```

Account for `joules` J being used. This is support in simgrid mode, to record how much energy was used, and will show up in further call to starpu_energy_used(). See Energy-based Scheduling fore more details.

57.42.3.22 starpu_energy_used()

```
double starpu_energy_used (
            void )
```

Return the amount of energy having been used in J. This account the amounts passed to starpu_energy_use(), but also the static energy use set by the STARPU_IDLE_POWER environment variable. See Energy-based Scheduling fore more details.

57.43 Task Bundles

Typedefs

- typedef struct _starpu_task_bundle ∗ starpu_task_bundle_t

Functions

- void starpu_task_bundle_create (starpu_task_bundle_t ∗bundle)
- int starpu_task_bundle_insert (starpu_task_bundle_t bundle, struct starpu_task ∗task)
- int starpu_task_bundle_remove (starpu_task_bundle_t bundle, struct starpu_task ∗task)
- void starpu_task_bundle_close (starpu_task_bundle_t bundle)
- double starpu_task_bundle_expected_length (starpu_task_bundle_t bundle, struct starpu_perfmodel_arch ∗arch, unsigned nimpl)
- double starpu_task_bundle_expected_data_transfer_time (starpu_task_bundle_t bundle, unsigned memory_node)
- double starpu_task_bundle_expected_energy (starpu_task_bundle_t bundle, struct starpu_perfmodel_arch ∗arch, unsigned nimpl)

57.43.1 Detailed Description

57.43.2 Typedef Documentation

57.43.2.1 starpu_task_bundle_t

```
typedef struct _starpu_task_bundle* starpu_task_bundle_t
```

Opaque structure describing a list of tasks that should be scheduled on the same worker whenever it's possible. It must be considered as a hint given to the scheduler as there is no guarantee that they will be executed on the same worker.

57.43.3 Function Documentation

57.43.3.1 starpu_task_bundle_create()

```
void starpu_task_bundle_create (
            starpu_task_bundle_t * bundle )
```

Factory function creating and initializing `bundle`, when the call returns, memory needed is allocated and `bundle` is ready to use.

57.43.3.2 starpu_task_bundle_insert()

```
int starpu_task_bundle_insert (
            starpu_task_bundle_t bundle,
            struct starpu_task * task )
```

Insert `task` in `bundle`. Until `task` is removed from `bundle` its expected length and data transfer time will be considered along those of the other tasks of bundle. This function must not be called if `bundle` is already closed and/or `task` is already submitted. On success, it returns 0. There are two cases of error : if `bundle` is already closed it returns `-EPERM`, if `task` was already submitted it returns `-EINVAL`.

57.43.3.3 starpu_task_bundle_remove()

```
int starpu_task_bundle_remove (
            starpu_task_bundle_t bundle,
            struct starpu_task * task )
```

Remove `task` from `bundle`. Of course `task` must have been previously inserted in `bundle`. This function must not be called if `bundle` is already closed and/or `task` is already submitted. Doing so would result in undefined behaviour. On success, it returns 0. If `bundle` is already closed it returns `-ENOENT`.

57.43.3.4 starpu_task_bundle_close()

```
void starpu_task_bundle_close (
            starpu_task_bundle_t bundle )
```

Inform the runtime that the user will not modify `bundle` anymore, it means no more inserting or removing task. Thus the runtime can destroy it when possible.

57.43.3.5 starpu_task_bundle_expected_length()

```
double starpu_task_bundle_expected_length (
            starpu_task_bundle_t bundle,
            struct starpu_perfmodel_arch * arch,
            unsigned nimpl )
```

Return the expected duration of `bundle` in micro-seconds.

57.43.3.6 starpu_task_bundle_expected_data_transfer_time()

```
double starpu_task_bundle_expected_data_transfer_time (
            starpu_task_bundle_t bundle,
            unsigned memory_node )
```

Return the time (in micro-seconds) expected to transfer all data used within `bundle`.

57.43.3.7 starpu_task_bundle_expected_energy()

```
double starpu_task_bundle_expected_energy (
            starpu_task_bundle_t bundle,
            struct starpu_perfmodel_arch * arch,
            unsigned nimpl )
```

Return the expected energy consumption of `bundle` in J.

57.44 Task Lists

Data Structures

- struct starpu_task_list

Functions

- void starpu_task_list_init (struct starpu_task_list ∗list)
- void starpu_task_list_push_front (struct starpu_task_list ∗list, struct starpu_task ∗task)
- void starpu_task_list_push_back (struct starpu_task_list ∗list, struct starpu_task ∗task)
- struct starpu_task ∗ starpu_task_list_front (const struct starpu_task_list ∗list)
- struct starpu_task ∗ starpu_task_list_back (const struct starpu_task_list ∗list)
- int starpu_task_list_empty (const struct starpu_task_list ∗list)
- void starpu_task_list_erase (struct starpu_task_list ∗list, struct starpu_task ∗task)
- struct starpu_task ∗ starpu_task_list_pop_front (struct starpu_task_list ∗list)
- struct starpu_task ∗ starpu_task_list_pop_back (struct starpu_task_list ∗list)
- struct starpu_task ∗ starpu_task_list_begin (const struct starpu_task_list ∗list)
- struct starpu_task ∗ starpu_task_list_end (const struct starpu_task_list ∗list STARPU_ATTRIBUTE_UNUSED)
- struct starpu_task ∗ starpu_task_list_next (const struct starpu_task ∗task)
- int starpu_task_list_ismember (const struct starpu_task_list ∗list, const struct starpu_task ∗look)
- void starpu_task_list_move (struct starpu_task_list ∗ldst, struct starpu_task_list ∗lsrc)

57.44.1 Detailed Description

57.44.2 Data Structure Documentation

57.44.2.1 struct starpu_task_list

Store a double-chained list of tasks

Data Fields

struct starpu_task ∗	head	head of the list
struct starpu_task ∗	tail	tail of the list

57.44.3 Function Documentation

57.44.3.1 starpu_task_list_init()

```
void starpu_task_list_init (
            struct starpu_task_list * list )
```

Initialize a list structure. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.44.3.2 starpu_task_list_push_front()

```
void starpu_task_list_push_front (
            struct starpu_task_list * list,
            struct starpu_task * task )
```

Push `task` at the front of `list`. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.44.3.3 starpu_task_list_push_back()

```
void starpu_task_list_push_back (
            struct starpu_task_list * list,
            struct starpu_task * task )
```

Push `task` at the back of `list`. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.44.3.4 starpu_task_list_front()

```
struct starpu_task * starpu_task_list_front (
            const struct starpu_task_list * list )
```

Get the front of `list` (without removing it). See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.44.3.5 starpu_task_list_back()

```
struct starpu_task * starpu_task_list_back (
            const struct starpu_task_list * list )
```

Get the back of `list` (without removing it). See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.44.3.6 starpu_task_list_empty()

```
int starpu_task_list_empty (
            const struct starpu_task_list * list )
```

Test if `list` is empty. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.44.3.7 starpu_task_list_erase()

```
void starpu_task_list_erase (
            struct starpu_task_list * list,
            struct starpu_task * task )
```

Remove `task` from `list`. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.44.3.8 starpu_task_list_pop_front()

```
struct starpu_task * starpu_task_list_pop_front (
            struct starpu_task_list * list )
```

Remove the element at the front of `list`. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.44.3.9 starpu_task_list_pop_back()

```
struct starpu_task * starpu_task_list_pop_back (
            struct starpu_task_list * list )
```

Remove the element at the back of `list`. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.44.3.10 starpu_task_list_begin()

```
struct starpu_task * starpu_task_list_begin (
            const struct starpu_task_list * list )
```

Get the first task of `list`. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.44.3.11 starpu_task_list_end()

```
struct starpu_task * starpu_task_list_end (
            const struct starpu_task_list *list STARPU_ATTRIBUTE_UNUSED )
```

Get the end of `list`. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.44.3.12 starpu_task_list_next()

```
struct starpu_task * starpu_task_list_next (
             const struct starpu_task * task )
```

Get the next task of `list`. This is not erase-safe. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.44.3.13 starpu_task_list_ismember()

```
int starpu_task_list_ismember (
             const struct starpu_task_list * list,
             const struct starpu_task * look )
```

Test whether the given task `look` is contained in the `list`. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.44.3.14 starpu_task_list_move()

```
void starpu_task_list_move (
             struct starpu_task_list * ldst,
             struct starpu_task_list * lsrc )
```

Move list from one head `lsrc` to another `ldst`. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.45 Theoretical Lower Bound on Execution Time

Compute theoretical upper computation efficiency bound corresponding to some actual execution.

Functions

- void starpu_bound_start (int deps, int prio)
- void starpu_bound_stop (void)
- void starpu_bound_print_dot (FILE ∗output)
- void starpu_bound_compute (double ∗res, double ∗integer_res, int integer)
- void starpu_bound_print_lp (FILE ∗output)
- void starpu_bound_print_mps (FILE ∗output)
- void starpu_bound_print (FILE ∗output, int integer)

57.45.1 Detailed Description

Compute theoretical upper computation efficiency bound corresponding to some actual execution.

57.45.2 Function Documentation

57.45.2.1 starpu_bound_start()

```
void starpu_bound_start (
             int deps,
             int prio )
```

Start recording tasks (resets stats). `deps` tells whether dependencies should be recorded too (this is quite expensive)
See Theoretical Lower Bound On Execution Time for more details.

57.45.2.2 starpu_bound_stop()

```
void starpu_bound_stop (
             void   )
```

Stop recording tasks
See Theoretical Lower Bound On Execution Time for more details.

57.45.2.3 starpu_bound_print_dot()

```
void starpu_bound_print_dot (
             FILE * output )
```

Emit the DAG that was recorded on `output`.
See Theoretical Lower Bound On Execution Time for more details.

57.45.2.4 starpu_bound_compute()

```
void starpu_bound_compute (
             double * res,
             double * integer_res,
             int integer )
```

Get theoretical upper bound (in ms) (needs glpk support detected by configure script). It returns 0 if some performance models are not calibrated. `integer` permits to choose between integer solving (which takes a long time but is correct), and relaxed solving (which provides an approximate solution).
See Theoretical Lower Bound On Execution Time for more details.

57.45.2.5 starpu_bound_print_lp()

```
void starpu_bound_print_lp (
            FILE * output )
```

Emit the Linear Programming system on `output` for the recorded tasks, in the lp format
See Theoretical Lower Bound On Execution Time for more details.

57.45.2.6 starpu_bound_print_mps()

```
void starpu_bound_print_mps (
            FILE * output )
```

Emit the Linear Programming system on `output` for the recorded tasks, in the mps format
See Theoretical Lower Bound On Execution Time for more details.

57.45.2.7 starpu_bound_print()

```
void starpu_bound_print (
            FILE * output,
            int integer )
```

Emit on `output` the statistics of actual execution vs theoretical upper bound. `integer` permits to choose between integer solving (which takes a long time but is correct), and relaxed solving (which provides an approximate solution).
See Theoretical Lower Bound On Execution Time for more details.

57.46 Threads

API for thread. The thread functions are either implemented on top of the pthread library or the SimGrid library when the simulated performance mode is enabled (SimGrid Support).

Macros

- #define STARPU_PTHREAD_CREATE_ON(name, thread, attr, routine, arg, where)
- #define STARPU_PTHREAD_CREATE(thread, attr, routine, arg)
- #define STARPU_PTHREAD_MUTEX_INIT(mutex, attr)
- #define STARPU_PTHREAD_MUTEX_INIT0(mutex, attr)
- #define STARPU_PTHREAD_MUTEX_DESTROY(mutex)
- #define STARPU_PTHREAD_MUTEX_LOCK(mutex)
- #define STARPU_PTHREAD_MUTEX_UNLOCK(mutex)
- #define STARPU_PTHREAD_KEY_CREATE(key, destr)
- #define STARPU_PTHREAD_KEY_DELETE(key)
- #define STARPU_PTHREAD_SETSPECIFIC(key, ptr)
- #define STARPU_PTHREAD_GETSPECIFIC(key)
- #define STARPU_PTHREAD_RWLOCK_INIT(rwlock, attr)
- #define STARPU_PTHREAD_RWLOCK_INIT0(rwlock, attr)
- #define STARPU_PTHREAD_RWLOCK_RDLOCK(rwlock)
- #define STARPU_PTHREAD_RWLOCK_WRLOCK(rwlock)
- #define STARPU_PTHREAD_RWLOCK_UNLOCK(rwlock)
- #define STARPU_PTHREAD_RWLOCK_DESTROY(rwlock)
- #define STARPU_PTHREAD_COND_INIT(cond, attr)
- #define STARPU_PTHREAD_COND_INIT0(cond, attr)
- #define STARPU_PTHREAD_COND_DESTROY(cond)
- #define STARPU_PTHREAD_COND_SIGNAL(cond)
- #define STARPU_PTHREAD_COND_BROADCAST(cond)
- #define STARPU_PTHREAD_COND_WAIT(cond, mutex)
- #define STARPU_PTHREAD_BARRIER_INIT(barrier, attr, count)
- #define STARPU_PTHREAD_BARRIER_DESTROY(barrier)
- #define STARPU_PTHREAD_BARRIER_WAIT(barrier)
- #define STARPU_PTHREAD_MUTEX_INITIALIZER
- #define STARPU_PTHREAD_COND_INITIALIZER

Functions

- int starpu_pthread_create (starpu_pthread_t ∗thread, const starpu_pthread_attr_t ∗attr, void ∗(∗start_routine)(void ∗), void ∗arg)
- int starpu_pthread_join (starpu_pthread_t thread, void ∗∗retval)
- int starpu_pthread_exit (void ∗retval) STARPU_ATTRIBUTE_NORETURN
- int starpu_pthread_attr_init (starpu_pthread_attr_t ∗attr)
- int starpu_pthread_attr_destroy (starpu_pthread_attr_t ∗attr)
- int starpu_pthread_attr_setdetachstate (starpu_pthread_attr_t ∗attr, int detachstate)
- int starpu_pthread_mutex_init (starpu_pthread_mutex_t ∗mutex, const starpu_pthread_mutexattr_t ∗mutexattr)
- int starpu_pthread_mutex_destroy (starpu_pthread_mutex_t ∗mutex)
- int starpu_pthread_mutex_lock (starpu_pthread_mutex_t ∗mutex)
- int starpu_pthread_mutex_unlock (starpu_pthread_mutex_t ∗mutex)
- int starpu_pthread_mutex_trylock (starpu_pthread_mutex_t ∗mutex)
- int starpu_pthread_mutexattr_gettype (const starpu_pthread_mutexattr_t ∗attr, int ∗type)
- int starpu_pthread_mutexattr_settype (starpu_pthread_mutexattr_t ∗attr, int type)
- int starpu_pthread_mutexattr_destroy (starpu_pthread_mutexattr_t ∗attr)
- int starpu_pthread_mutexattr_init (starpu_pthread_mutexattr_t ∗attr)

- int starpu_pthread_key_create (starpu_pthread_key_t ∗key, void(∗destr_function)(void ∗))
- int starpu_pthread_key_delete (starpu_pthread_key_t key)
- int starpu_pthread_setspecific (starpu_pthread_key_t key, const void ∗pointer)
- void ∗ starpu_pthread_getspecific (starpu_pthread_key_t key)
- int starpu_pthread_cond_init (starpu_pthread_cond_t ∗cond, starpu_pthread_condattr_t ∗cond_attr)
- int starpu_pthread_cond_signal (starpu_pthread_cond_t ∗cond)
- int starpu_pthread_cond_broadcast (starpu_pthread_cond_t ∗cond)
- int starpu_pthread_cond_wait (starpu_pthread_cond_t ∗cond, starpu_pthread_mutex_t ∗mutex)
- int starpu_pthread_cond_timedwait (starpu_pthread_cond_t ∗cond, starpu_pthread_mutex_t ∗mutex, const struct timespec ∗abstime)
- int starpu_pthread_cond_destroy (starpu_pthread_cond_t ∗cond)
- int starpu_pthread_rwlock_init (starpu_pthread_rwlock_t ∗rwlock, const starpu_pthread_rwlockattr_t ∗attr)
- int starpu_pthread_rwlock_destroy (starpu_pthread_rwlock_t ∗rwlock)
- int starpu_pthread_rwlock_rdlock (starpu_pthread_rwlock_t ∗rwlock)
- int starpu_pthread_rwlock_tryrdlock (starpu_pthread_rwlock_t ∗rwlock)
- int starpu_pthread_rwlock_wrlock (starpu_pthread_rwlock_t ∗rwlock)
- int starpu_pthread_rwlock_trywrlock (starpu_pthread_rwlock_t ∗rwlock)
- int starpu_pthread_rwlock_unlock (starpu_pthread_rwlock_t ∗rwlock)
- int starpu_pthread_barrier_init (starpu_pthread_barrier_t ∗barrier, const starpu_pthread_barrierattr_t ∗attr, unsigned count)
- int starpu_pthread_barrier_destroy (starpu_pthread_barrier_t ∗barrier)
- int starpu_pthread_barrier_wait (starpu_pthread_barrier_t ∗barrier)
- int starpu_pthread_spin_init (starpu_pthread_spinlock_t ∗lock, int pshared)
- int starpu_pthread_spin_destroy (starpu_pthread_spinlock_t ∗lock)
- int starpu_pthread_spin_lock (starpu_pthread_spinlock_t ∗lock)
- int starpu_pthread_spin_trylock (starpu_pthread_spinlock_t ∗lock)
- int starpu_pthread_spin_unlock (starpu_pthread_spinlock_t ∗lock)

57.46.1 Detailed Description

API for thread. The thread functions are either implemented on top of the pthread library or the SimGrid library when the simulated performance mode is enabled (SimGrid Support).

57.46.2 Macro Definition Documentation

57.46.2.1 STARPU_PTHREAD_CREATE_ON

```
#define STARPU_PTHREAD_CREATE_ON(
              name,
              thread,
              attr,
              routine,
              arg,
              where )
```

Call starpu_pthread_create_on() and abort on error.

57.46.2.2 STARPU_PTHREAD_CREATE

```
#define STARPU_PTHREAD_CREATE(
              thread,
              attr,
              routine,
              arg )
```

Call starpu_pthread_create() and abort on error.

57.46.2.3 STARPU_PTHREAD_MUTEX_INIT

```
#define STARPU_PTHREAD_MUTEX_INIT(
              mutex,
              attr )
```

Call starpu_pthread_mutex_init() and abort on error.

57.46.2.4 STARPU_PTHREAD_MUTEX_INIT0

```
#define STARPU_PTHREAD_MUTEX_INIT0(
              mutex,
              attr )
```

Call starpu_pthread_mutex_init() only if the content of PTHREAD_MUTEX_INITIALIZER is not zero. This should be called instead of STARPU_PTHREAD_MUTEX_INIT when it is known that the content of the pthread_mutex_t was already zeroed.

57.46.2.5 STARPU_PTHREAD_MUTEX_DESTROY

```
#define STARPU_PTHREAD_MUTEX_DESTROY(
              mutex )
```

Call starpu_pthread_mutex_destroy() and abort on error.

57.46.2.6 STARPU_PTHREAD_MUTEX_LOCK

```
#define STARPU_PTHREAD_MUTEX_LOCK(
              mutex )
```

Call starpu_pthread_mutex_lock() and abort on error.

57.46.2.7 STARPU_PTHREAD_MUTEX_UNLOCK

```
#define STARPU_PTHREAD_MUTEX_UNLOCK(
              mutex )
```

Call starpu_pthread_mutex_unlock() and abort on error.

57.46.2.8 STARPU_PTHREAD_KEY_CREATE

```
#define STARPU_PTHREAD_KEY_CREATE(
              key,
              destr )
```

Call starpu_pthread_key_create() and abort on error.

57.46.2.9 STARPU_PTHREAD_KEY_DELETE

```
#define STARPU_PTHREAD_KEY_DELETE(
              key )
```

Call starpu_pthread_key_delete() and abort on error.

57.46.2.10 STARPU_PTHREAD_SETSPECIFIC

```
#define STARPU_PTHREAD_SETSPECIFIC(
              key,
              ptr )
```

Call starpu_pthread_setspecific() and abort on error.

57.46.2.11 STARPU_PTHREAD_GETSPECIFIC

```
#define STARPU_PTHREAD_GETSPECIFIC(
              key )
```

Call starpu_pthread_getspecific() and abort on error.

57.46.2.12 STARPU_PTHREAD_RWLOCK_INIT

```
#define STARPU_PTHREAD_RWLOCK_INIT(
              rwlock,
              attr )
```

Call starpu_pthread_rwlock_init() and abort on error.

57.46.2.13 STARPU_PTHREAD_RWLOCK_INIT0

```
#define STARPU_PTHREAD_RWLOCK_INIT0(
              rwlock,
              attr )
```

Call starpu_pthread_rwlock_init() only if the content of PTHREAD_RWLOCK_INITIALIZER is not zero. This should be called instead of STARPU_PTHREAD_RWLOCK_INIT when it is known that the content of the pthread_rwlock_t was already zeroed.

57.46.2.14 STARPU_PTHREAD_RWLOCK_RDLOCK

```
#define STARPU_PTHREAD_RWLOCK_RDLOCK(
              rwlock )
```

Call starpu_pthread_rwlock_rdlock() and abort on error.

57.46.2.15 STARPU_PTHREAD_RWLOCK_WRLOCK

```
#define STARPU_PTHREAD_RWLOCK_WRLOCK(
              rwlock )
```

Call starpu_pthread_rwlock_wrlock() and abort on error.

57.46.2.16 STARPU_PTHREAD_RWLOCK_UNLOCK

```
#define STARPU_PTHREAD_RWLOCK_UNLOCK(
              rwlock )
```

Call starpu_pthread_rwlock_unlock() and abort on error.

57.46.2.17 STARPU_PTHREAD_RWLOCK_DESTROY

```
#define STARPU_PTHREAD_RWLOCK_DESTROY(
              rwlock )
```

Call starpu_pthread_rwlock_destroy() and abort on error.

57.46.2.18 STARPU_PTHREAD_COND_INIT

```
#define STARPU_PTHREAD_COND_INIT(
              cond,
              attr )
```

Call starpu_pthread_cond_init() and abort on error.

57.46.2.19 STARPU_PTHREAD_COND_INIT0

```
#define STARPU_PTHREAD_COND_INIT0(
              cond,
              attr )
```

Call starpu_pthread_cond_init() only if the content of PTHREAD_COND_INITIALIZER is not zero. This should be called instead of STARPU_PTHREAD_COND_INIT when it is known that the content of the pthread_cond_t was already zeroed.

57.46.2.20 STARPU_PTHREAD_COND_DESTROY

```
#define STARPU_PTHREAD_COND_DESTROY(
              cond )
```

Call starpu_pthread_cond_destroy() and abort on error.

57.46.2.21 STARPU_PTHREAD_COND_SIGNAL

```
#define STARPU_PTHREAD_COND_SIGNAL(
              cond )
```

Call starpu_pthread_cond_signal() and abort on error.

57.46.2.22 STARPU_PTHREAD_COND_BROADCAST

```
#define STARPU_PTHREAD_COND_BROADCAST(
              cond )
```

Call starpu_pthread_cond_broadcast() and abort on error.

57.46.2.23 STARPU_PTHREAD_COND_WAIT

```
#define STARPU_PTHREAD_COND_WAIT(
              cond,
              mutex )
```

Call starpu_pthread_cond_wait() and abort on error.

57.46.2.24 STARPU_PTHREAD_BARRIER_INIT

```
#define STARPU_PTHREAD_BARRIER_INIT(
              barrier,
              attr,
              count )
```

Call starpu_pthread_barrier_init() and abort on error.

57.46.2.25 STARPU_PTHREAD_BARRIER_DESTROY

```
#define STARPU_PTHREAD_BARRIER_DESTROY(
              barrier )
```

Call starpu_pthread_barrier_destroy() and abort on error.

57.46.2.26 STARPU_PTHREAD_BARRIER_WAIT

```
#define STARPU_PTHREAD_BARRIER_WAIT(
              barrier )
```

Call starpu_pthread_barrier_wait() and abort on error.

57.46.2.27 STARPU_PTHREAD_MUTEX_INITIALIZER

```
STARPU_PTHREAD_MUTEX_INITIALIZER
```

Initialize the mutex given in parameter.

57.46.2.28 STARPU_PTHREAD_COND_INITIALIZER

```
STARPU_PTHREAD_COND_INITIALIZER
```

Initialize the condition variable given in parameter.

57.46.3 Function Documentation

57.46.3.1 starpu_pthread_create()

```
int starpu_pthread_create (
            starpu_pthread_t * thread,
            const starpu_pthread_attr_t * attr,
            void *(*)(void *) start_routine,
            void * arg )
```

Start a new thread in the calling process. The new thread starts execution by invoking `start_routine`; `arg` is passed as the sole argument of `start_routine`.

57.46.3.2 starpu_pthread_join()

```
int starpu_pthread_join (
            starpu_pthread_t thread,
            void ** retval )
```

Wait for the thread specified by `thread` to terminate. If that thread has already terminated, then the function returns immediately. The thread specified by `thread` must be joinable.

57.46.3.3 starpu_pthread_exit()

```
int starpu_pthread_exit (
            void * retval )
```

Terminate the calling thread and return a value via `retval` that (if the thread is joinable) is available to another thread in the same process that calls starpu_pthread_join().

57.46.3.4 starpu_pthread_attr_init()

```
int starpu_pthread_attr_init (
            starpu_pthread_attr_t * attr )
```

Initialize the thread attributes object pointed to by `attr` with default attribute values.
Do not do anything when the simulated performance mode is enabled (SimGrid Support).

57.46.3.5 starpu_pthread_attr_destroy()

```
int starpu_pthread_attr_destroy (
            starpu_pthread_attr_t * attr )
```

Destroy a thread attributes object which is no longer required. Destroying a thread attributes object has no effect on threads that were created using that object.
Do not do anything when the simulated performance mode is enabled (SimGrid Support).

57.46.3.6 starpu_pthread_attr_setdetachstate()

```
int starpu_pthread_attr_setdetachstate (
            starpu_pthread_attr_t * attr,
            int detachstate )
```

Set the detach state attribute of the thread attributes object referred to by `attr` to the value specified in `detachstate`. The detach state attribute determines whether a thread created using the thread attributes object `attr` will be created in a joinable or a detached state.
Do not do anything when the simulated performance mode is enabled (SimGrid Support).

57.46.3.7 starpu_pthread_mutex_init()

```
int starpu_pthread_mutex_init (
            starpu_pthread_mutex_t * mutex,
            const starpu_pthread_mutexattr_t * mutexattr )
```

Initialize the mutex object pointed to by `mutex` according to the mutex attributes specified in `mutexattr`. If `mutexattr` is `NULL`, default attributes are used instead.

57.46.3.8 starpu_pthread_mutex_destroy()

```
int starpu_pthread_mutex_destroy (
            starpu_pthread_mutex_t * mutex )
```

Destroy a mutex object, and free the resources it might hold. The mutex must be unlocked on entrance.

57.46.3.9 starpu_pthread_mutex_lock()

```
int starpu_pthread_mutex_lock (
            starpu_pthread_mutex_t * mutex )
```

Lock the given `mutex`. If `mutex` is currently unlocked, it becomes locked and owned by the calling thread, and the function returns immediately. If `mutex` is already locked by another thread, the function suspends the calling thread until `mutex` is unlocked.
This function also produces trace when the configure option --enable-fxt-lock is enabled.

57.46.3.10 starpu_pthread_mutex_unlock()

```
int starpu_pthread_mutex_unlock (
            starpu_pthread_mutex_t * mutex )
```

Unlock the given `mutex`. The mutex is assumed to be locked and owned by the calling thread on entrance to starpu_pthread_mutex_unlock().
This function also produces trace when the configure option --enable-fxt-lock is enabled.

57.46.3.11 starpu_pthread_mutex_trylock()

```
int starpu_pthread_mutex_trylock (
            starpu_pthread_mutex_t * mutex )
```

Behave identically to starpu_pthread_mutex_lock(), except that it does not block the calling thread if the mutex is already locked by another thread (or by the calling thread in the case of a ``fast'' mutex). Instead, the function returns immediately with the error code `EBUSY`.
This function also produces trace when the configure option --enable-fxt-lock is enabled.

57.46.3.12 starpu_pthread_mutexattr_gettype()

```
int starpu_pthread_mutexattr_gettype (
            const starpu_pthread_mutexattr_t * attr,
            int * type )
```

todo

57.46.3.13 starpu_pthread_mutexattr_settype()

```
int starpu_pthread_mutexattr_settype (
            starpu_pthread_mutexattr_t * attr,
            int type )
```

todo

57.46.3.14 starpu_pthread_mutexattr_destroy()

```
int starpu_pthread_mutexattr_destroy (
            starpu_pthread_mutexattr_t * attr )
```

todo

57.46.3.15 starpu_pthread_mutexattr_init()

```
int starpu_pthread_mutexattr_init (
            starpu_pthread_mutexattr_t * attr )
```

todo

57.46.3.16 starpu_pthread_key_create()

```
int starpu_pthread_key_create (
            starpu_pthread_key_t * key,
            void(*)(void *) destr_function )
```

Allocate a new TSD key. The key is stored in the location pointed to by `key`.

57.46.3.17 starpu_pthread_key_delete()

```
int starpu_pthread_key_delete (
            starpu_pthread_key_t key )
```

Deallocate a TSD key. Do not check whether non-`NULL` values are associated with that key in the currently executing threads, nor call the destructor function associated with the key.

57.46.3.18 starpu_pthread_setspecific()

```
int starpu_pthread_setspecific (
            starpu_pthread_key_t key,
            const void * pointer )
```

Change the value associated with `key` in the calling thread, storing the given `pointer` instead.

57.46.3.19 starpu_pthread_getspecific()

```
void * starpu_pthread_getspecific (
            starpu_pthread_key_t key )
```

Return the value associated with `key` on success, and `NULL` on error.

57.46.3.20 starpu_pthread_cond_init()

```
int starpu_pthread_cond_init (
            starpu_pthread_cond_t * cond,
            starpu_pthread_condattr_t * cond_attr )
```

Initialize the condition variable `cond`, using the condition attributes specified in `cond_attr`, or default attributes if `cond_attr` is `NULL`.

57.46.3.21 starpu_pthread_cond_signal()

```
int starpu_pthread_cond_signal (
            starpu_pthread_cond_t * cond )
```

Restart one of the threads that are waiting on the condition variable `cond`. If no threads are waiting on `cond`, nothing happens. If several threads are waiting on `cond`, exactly one is restarted, but it is not specified which.

57.46.3.22 starpu_pthread_cond_broadcast()

```
int starpu_pthread_cond_broadcast (
            starpu_pthread_cond_t * cond )
```

Restart all the threads that are waiting on the condition variable `cond`. Nothing happens if no threads are waiting on `cond`.

57.46.3.23 starpu_pthread_cond_wait()

```
int starpu_pthread_cond_wait (
            starpu_pthread_cond_t * cond,
            starpu_pthread_mutex_t * mutex )
```

Atomically unlock `mutex` (as per starpu_pthread_mutex_unlock()) and wait for the condition variable `cond` to be signaled. The thread execution is suspended and does not consume any CPU time until the condition variable is signaled. The mutex must be locked by the calling thread on entrance to starpu_pthread_cond_wait(). Before returning to the calling thread, the function re-acquires mutex (as per starpu_pthread_mutex_lock()).
This function also produces trace when the configure option --enable-fxt-lock is enabled.

57.46.3.24 starpu_pthread_cond_timedwait()

```
int starpu_pthread_cond_timedwait (
            starpu_pthread_cond_t * cond,
            starpu_pthread_mutex_t * mutex,
            const struct timespec * abstime )
```

Atomicall unlocks `mutex` and wait on `cond`, as starpu_pthread_cond_wait() does, but also bound the duration of the wait with `abstime`.

57.46.3.25 starpu_pthread_cond_destroy()

```
int starpu_pthread_cond_destroy (
            starpu_pthread_cond_t * cond )
```

Destroy a condition variable, freeing the resources it might hold. No threads must be waiting on the condition variable on entrance to the function.

57.46.3.26 starpu_pthread_rwlock_init()

```
int starpu_pthread_rwlock_init (
            starpu_pthread_rwlock_t * rwlock,
            const starpu_pthread_rwlockattr_t * attr )
```

Similar to starpu_pthread_mutex_init().

57.46.3.27 starpu_pthread_rwlock_destroy()

```
int starpu_pthread_rwlock_destroy (
            starpu_pthread_rwlock_t * rwlock )
```

Similar to starpu_pthread_mutex_destroy().

57.46.3.28 starpu_pthread_rwlock_rdlock()

```
int starpu_pthread_rwlock_rdlock (
            starpu_pthread_rwlock_t * rwlock )
```

Similar to starpu_pthread_mutex_lock().

57.46.3.29 starpu_pthread_rwlock_tryrdlock()

```
int starpu_pthread_rwlock_tryrdlock (
            starpu_pthread_rwlock_t * rwlock )
```

todo

57.46.3.30 starpu_pthread_rwlock_wrlock()

```
int starpu_pthread_rwlock_wrlock (
            starpu_pthread_rwlock_t * rwlock )
```

Similar to starpu_pthread_mutex_lock().

57.46.3.31 starpu_pthread_rwlock_trywrlock()

```
int starpu_pthread_rwlock_trywrlock (
            starpu_pthread_rwlock_t * rwlock )
```

todo

57.46.3.32 starpu_pthread_rwlock_unlock()

```
int starpu_pthread_rwlock_unlock (
            starpu_pthread_rwlock_t * rwlock )
```

Similar to starpu_pthread_mutex_unlock().

57.46.3.33 starpu_pthread_barrier_init()

```
int starpu_pthread_barrier_init (
            starpu_pthread_barrier_t * barrier,
            const starpu_pthread_barrierattr_t * attr,
            unsigned count )
```

todo

57.46.3.34 starpu_pthread_barrier_destroy()

```
int starpu_pthread_barrier_destroy (
            starpu_pthread_barrier_t * barrier )
```

todo

57.46.3.35 starpu_pthread_barrier_wait()

```
int starpu_pthread_barrier_wait (
            starpu_pthread_barrier_t * barrier )
```

todo

57.46.3.36 starpu_pthread_spin_init()

```
int starpu_pthread_spin_init (
            starpu_pthread_spinlock_t * lock,
            int pshared )
```

todo

57.46.3.37 starpu_pthread_spin_destroy()

```
int starpu_pthread_spin_destroy (
            starpu_pthread_spinlock_t * lock )
```

todo

57.46.3.38 starpu_pthread_spin_lock()

```
int starpu_pthread_spin_lock (
            starpu_pthread_spinlock_t * lock )
```

todo

57.46.3.39 starpu_pthread_spin_trylock()

```
int starpu_pthread_spin_trylock (
            starpu_pthread_spinlock_t * lock )
```

todo

57.46.3.40 starpu_pthread_spin_unlock()

```
int starpu_pthread_spin_unlock (
            starpu_pthread_spinlock_t * lock )
```

todo

57.47 Toolbox

The following macros allow to make GCC extensions portable, and to have a code which can be compiled with any C compiler.

Macros

- #define STARPU_GNUC_PREREQ(maj, min)
- #define STARPU_UNLIKELY(expr)
- #define STARPU_LIKELY(expr)
- #define STARPU_ATTRIBUTE_UNUSED
- #define STARPU_ATTRIBUTE_NORETURN
- #define STARPU_ATTRIBUTE_VISIBILITY_DEFAULT
- #define STARPU_VISIBILITY_PUSH_HIDDEN
- #define STARPU_VISIBILITY_POP
- #define STARPU_ATTRIBUTE_MALLOC
- #define STARPU_ATTRIBUTE_WARN_UNUSED_RESULT
- #define STARPU_ATTRIBUTE_PURE
- #define STARPU_ATTRIBUTE_ALIGNED(size)
- #define **STARPU_ATTRIBUTE_FORMAT**(type, string, first)
- #define **STARPU_INLINE**
- #define **STARPU_ATTRIBUTE_CALLOC_SIZE**(num, size)
- #define **STARPU_ATTRIBUTE_ALLOC_SIZE**(size)
- #define **STARPU_BACKTRACE_LENGTH**
- #define **STARPU_DUMP_BACKTRACE**()
- #define **STARPU_SIMGRID_ASSERT**(x)
- #define STARPU_ASSERT(x)
- #define STARPU_ASSERT_ACCESSIBLE(ptr)
- #define STARPU_STATIC_ASSERT(x)
- #define STARPU_ASSERT_MSG(x, msg, ...)
- #define **_starpu_abort**()
- #define STARPU_ABORT()
- #define STARPU_ABORT_MSG(msg, ...)
- #define STARPU_CHECK_RETURN_VALUE(err, message, ...)
- #define STARPU_CHECK_RETURN_VALUE_IS(err, value, message, ...)
- #define **STARPU_ATOMIC_SOMETHING**(name, expr)
- #define **STARPU_ATOMIC_SOMETHINGL**(name, expr)
- #define **STARPU_ATOMIC_SOMETHING64**(name, expr)
- #define **STARPU_BOOL_COMPARE_AND_SWAP_PTR**(ptr, old, value)
- #define **STARPU_VAL_COMPARE_AND_SWAP_PTR**(ptr, old, value)
- #define STARPU_RMB()
- #define STARPU_WMB()
- #define **STARPU_CACHELINE_SIZE**

57.47.1 Detailed Description

The following macros allow to make GCC extensions portable, and to have a code which can be compiled with any C compiler.

57.47.2 Macro Definition Documentation

57.47.2.1 STARPU_GNUC_PREREQ

```
#define STARPU_GNUC_PREREQ(
            maj,
            min )
```

Return true (non-zero) if GCC version `maj.min` or later is being used (macro taken from glibc.)

57.47.2.2 STARPU_UNLIKELY

```
#define STARPU_UNLIKELY(
            expr )
```

When building with a GNU C Compiler, allow programmers to mark an expression as unlikely.

57.47.2.3 STARPU_LIKELY

```
#define STARPU_LIKELY(
            expr )
```

When building with a GNU C Compiler, allow programmers to mark an expression as likely.

57.47.2.4 STARPU_ATTRIBUTE_UNUSED

```
#define STARPU_ATTRIBUTE_UNUSED
```

When building with a GNU C Compiler, defined to **attribute**((unused))

57.47.2.5 STARPU_ATTRIBUTE_NORETURN

```
#define STARPU_ATTRIBUTE_NORETURN
```

When building with a GNU C Compiler, defined to **attribute**((noreturn))

57.47.2.6 STARPU_ATTRIBUTE_VISIBILITY_DEFAULT

```
#define STARPU_ATTRIBUTE_VISIBILITY_DEFAULT
```

When building with a GNU C Compiler, defined to **attribute**((visibility ("default")))

57.47.2.7 STARPU_VISIBILITY_PUSH_HIDDEN

```
#define STARPU_VISIBILITY_PUSH_HIDDEN
```

When building with a GNU C Compiler, defined to #pragma GCC visibility push(hidden)

57.47.2.8 STARPU_VISIBILITY_POP

```
#define STARPU_VISIBILITY_POP
```

When building with a GNU C Compiler, defined to #pragma GCC visibility pop

57.47.2.9 STARPU_ATTRIBUTE_MALLOC

```
#define STARPU_ATTRIBUTE_MALLOC
```

When building with a GNU C Compiler, defined to **attribute**((malloc))

57.47.2.10 STARPU_ATTRIBUTE_WARN_UNUSED_RESULT

```
#define STARPU_ATTRIBUTE_WARN_UNUSED_RESULT
```

When building with a GNU C Compiler, defined to **attribute**((warn_unused_result))

57.47.2.11 STARPU_ATTRIBUTE_PURE

```
#define STARPU_ATTRIBUTE_PURE
```

When building with a GNU C Compiler, defined to **attribute**((pure))

57.47.2.12 STARPU_ATTRIBUTE_ALIGNED

```
#define STARPU_ATTRIBUTE_ALIGNED(
                size )
```

When building with a GNU C Compiler, defined to__attribute__((aligned(size)))

57.47.2.13 STARPU_ASSERT

```
#define STARPU_ASSERT(
                x )
```

Unless StarPU has been configured with the option --enable-fast, this macro will abort if the expression `x` is false.

57.47.2.14 STARPU_ASSERT_ACCESSIBLE

```
#define STARPU_ASSERT_ACCESSIBLE(
                ptr )
```

Unless StarPU has been configured with the option --enable-fast, this macro will abort if the pointer `x` is not pointing to valid memory.

57.47.2.15 STARPU_STATIC_ASSERT

```
#define STARPU_STATIC_ASSERT(
                x )
```

This macro will abort compilation if the expression `x` is false.

57.47.2.16 STARPU_ASSERT_MSG

```
#define STARPU_ASSERT_MSG(
                x,
                msg,
                ... )
```

Unless StarPU has been configured with the option --enable-fast, this macro will abort if the expression `x` is false. The string `msg` will be displayed.

57.47.2.17 STARPU_ABORT

```
#define STARPU_ABORT( )
```

Abort the program.

57.47.2.18 STARPU_ABORT_MSG

```
#define STARPU_ABORT_MSG(
                msg,
                ... )
```

Print the string '[starpu][abort][name of the calling function:name of the `file:line` in the file]' followed by the given string `msg` and abort the program

57.47.2.19 STARPU_CHECK_RETURN_VALUE

```
#define STARPU_CHECK_RETURN_VALUE(
                err,
                message,
                ... )
```

Abort the program (after displaying `message`) if `err` has a value which is not 0.

57.47.2.20 STARPU_CHECK_RETURN_VALUE_IS

```
#define STARPU_CHECK_RETURN_VALUE_IS(
                err,
                value,
                message,
                ... )
```

Abort the program (after displaying `message`) if `err` is different from `value`.

57.47.2.21 STARPU_RMB

```
#define STARPU_RMB( )
```

This macro can be used to do a synchronization.

57.47.2.22 STARPU_WMB

```
#define STARPU_WMB( )
```

This macro can be used to do a synchronization.

57.48 Transactions

Functions

- struct starpu_transaction ∗ starpu_transaction_open (int(∗do_start_func)(void ∗buffer, void ∗arg), void ∗do_start_arg)
- void starpu_transaction_next_epoch (struct starpu_transaction ∗p_trs, void ∗do_start_arg)
- void starpu_transaction_close (struct starpu_transaction ∗p_trs)

57.48.1 Detailed Description

57.48.2 Function Documentation

57.48.2.1 starpu_transaction_open()

```
struct starpu_transaction * starpu_transaction_open (
            int(*)(void *buffer, void *arg) do_start_func,
            void * do_start_arg )
```

Function to open a new transaction object and start the first transaction epoch.

Returns

> A pointer to an initializes `struct starpu_transaction` or `NULL` if submitting the transaction begin task failed with `ENODEV`. See Transaction Creation for more details.

57.48.2.2 starpu_transaction_next_epoch()

```
void starpu_transaction_next_epoch (
            struct starpu_transaction * p_trs,
            void * do_start_arg )
```

Function to mark the end of the current transaction epoch and start a new epoch. See Epoch Transition for more details.

57.48.2.3 starpu_transaction_close()

```
void starpu_transaction_close (
            struct starpu_transaction * p_trs )
```

Function to mark the end of the last transaction epoch and free the transaction object. See Transaction Closing for more details.

57.49 Tree

API tree facilities.

Data Structures

- struct starpu_tree

Functions

- void **starpu_tree_reset_visited** (struct starpu_tree ∗tree, char ∗visited)
- void **starpu_tree_prepare_children** (unsigned arity, struct starpu_tree ∗father)
- void **starpu_tree_insert** (struct starpu_tree ∗tree, int id, int level, int is_pu, int arity, struct starpu_tree ∗father)
- struct starpu_tree ∗ **starpu_tree_get** (struct starpu_tree ∗tree, int id)
- struct starpu_tree ∗ **starpu_tree_get_neighbour** (struct starpu_tree ∗tree, struct starpu_tree ∗node, char ∗visited, char ∗present)
- void **starpu_tree_free** (struct starpu_tree ∗tree)

57.49.1 Detailed Description

API tree facilities.

57.49.2 Data Structure Documentation

57.49.2.1 struct starpu_tree

todo

Data Fields

struct starpu_tree ∗	nodes	
struct starpu_tree ∗	father	
int	arity	
int	id	
int	level	
int	is_pu	

57.50 Versioning

Macros

- #define STARPU_MAJOR_VERSION
- #define STARPU_MINOR_VERSION
- #define STARPU_RELEASE_VERSION

Functions

- void starpu_get_version (int ∗major, int ∗minor, int ∗release)

57.50.1 Detailed Description

57.50.2 Macro Definition Documentation

57.50.2.1 STARPU_MAJOR_VERSION

`#define STARPU_MAJOR_VERSION`

Define the major version of StarPU. This is the version used when compiling the application.

57.50.2.2 STARPU_MINOR_VERSION

`#define STARPU_MINOR_VERSION`

Define the minor version of StarPU. This is the version used when compiling the application.

57.50.2.3 STARPU_RELEASE_VERSION

`#define STARPU_RELEASE_VERSION`

Define the release version of StarPU. This is the version used when compiling the application.

57.50.3 Function Documentation

57.50.3.1 starpu_get_version()

```
void starpu_get_version (
             int * major,
             int * minor,
             int * release )
```

Return as 3 integers the version of StarPU used when running the application. See Configuration and Initialization for more details.

57.51 Workers

Data Structures

- struct starpu_sched_ctx_iterator
- struct starpu_worker_collection

Macros

- #define starpu_worker_get_id_check()
- #define STARPU_MAXNODES
- #define STARPU_MAXCPUS
- #define STARPU_MAXNUMANODES
- #define STARPU_NMAXWORKERS
- #define STARPU_UNKNOWN_WORKER

Enumerations

- enum starpu_node_kind {
  STARPU_UNUSED , STARPU_CPU_RAM , STARPU_CUDA_RAM , STARPU_OPENCL_RAM , STARPU_MAX_FPGA_RAM , STARPU_DISK_RAM , STARPU_MPI_MS_RAM , STARPU_TCPIP_MS_RAM ,
  STARPU_HIP_RAM , STARPU_MAX_RAM , STARPU_NRAM }
- enum starpu_worker_archtype {
  STARPU_CPU_WORKER , STARPU_CUDA_WORKER , STARPU_OPENCL_WORKER , STARPU_MAX_FPGA_WORKER ,
  STARPU_MPI_MS_WORKER , STARPU_TCPIP_MS_WORKER , STARPU_HIP_WORKER , STARPU_NARCH ,
  STARPU_ANY_WORKER }
- enum starpu_worker_collection_type { STARPU_WORKER_TREE , STARPU_WORKER_LIST }

Functions

- void starpu_worker_wait_for_initialisation (void)
- unsigned starpu_worker_archtype_is_valid (enum starpu_worker_archtype type)
- enum starpu_worker_archtype starpu_arch_mask_to_worker_archtype (unsigned mask)
- unsigned starpu_worker_get_count (void)
- unsigned starpu_cpu_worker_get_count (void)
- unsigned starpu_cuda_worker_get_count (void)
- unsigned starpu_hip_worker_get_count (void)
- unsigned starpu_opencl_worker_get_count (void)
- unsigned starpu_mpi_ms_worker_get_count (void)
- unsigned starpu_tcpip_ms_worker_get_count (void)
- int starpu_worker_get_id (void)
- unsigned **_starpu_worker_get_id_check** (const char ∗f, int l)
- int starpu_worker_get_bindid (int workerid)
- void starpu_sched_find_all_worker_combinations (void)
- enum starpu_worker_archtype starpu_worker_get_type (int id)
- int starpu_worker_get_count_by_type (enum starpu_worker_archtype type)
- unsigned starpu_worker_get_ids_by_type (enum starpu_worker_archtype type, int ∗workerids, unsigned maxsize)
- int starpu_worker_get_by_type (enum starpu_worker_archtype type, int num)
- int starpu_worker_get_by_devid (enum starpu_worker_archtype type, int devid)
- unsigned starpu_worker_type_can_execute_task (enum starpu_worker_archtype worker_type, const struct starpu_task ∗task)
- void starpu_worker_get_name (int id, char ∗dst, size_t maxlen)
- void starpu_worker_display_all (FILE ∗output)

- void starpu_worker_display_names (FILE ∗output, enum starpu_worker_archtype type)
- void starpu_worker_display_count (FILE ∗output, enum starpu_worker_archtype type)
- int starpu_worker_get_devid (int id)
- int starpu_worker_get_devnum (int id)
- int starpu_worker_get_subworkerid (int id)
- struct starpu_tree ∗ starpu_workers_get_tree (void)
- unsigned starpu_worker_get_sched_ctx_list (int worker, unsigned ∗∗sched_ctx)
- void starpu_worker_get_current_task_exp_end (unsigned workerid, struct timespec ∗date)
- unsigned starpu_worker_is_blocked_in_parallel (int workerid)
- unsigned starpu_worker_is_slave_somewhere (int workerid)
- const char ∗ starpu_worker_get_type_as_string (enum starpu_worker_archtype type)
- enum starpu_worker_archtype starpu_worker_get_type_from_string (const char ∗type)
- const char ∗ starpu_worker_get_type_as_env_var (enum starpu_worker_archtype type)
- int starpu_bindid_get_workerids (int bindid, int ∗∗workerids)
- int starpu_worker_get_devids (enum starpu_worker_archtype type, int ∗devids, int num)
- int starpu_worker_get_stream_workerids (unsigned devid, int ∗workerids, enum starpu_worker_archtype type)
- hwloc_cpuset_t starpu_worker_get_hwloc_cpuset (int workerid)
- hwloc_obj_t starpu_worker_get_hwloc_obj (int workerid)
- int starpu_memory_node_get_devid (unsigned node)
- unsigned starpu_worker_get_local_memory_node (void)
- unsigned starpu_worker_get_memory_node (unsigned workerid)
- unsigned starpu_memory_nodes_get_count (void)
- unsigned starpu_memory_nodes_get_count_by_kind (enum starpu_node_kind kind)
- unsigned starpu_memory_node_get_ids_by_type (enum starpu_node_kind kind, unsigned ∗memory_nodes_ids, unsigned maxsize)
- int starpu_memory_node_get_name (unsigned node, char ∗name, size_t size)
- unsigned starpu_memory_nodes_get_numa_count (void)
- int starpu_memory_nodes_numa_id_to_devid (int osid)
- int starpu_memory_nodes_numa_devid_to_id (unsigned id)
- enum starpu_node_kind starpu_node_get_kind (unsigned node)
- enum starpu_worker_archtype starpu_memory_node_get_worker_archtype (enum starpu_node_kind node_kind)
- enum starpu_node_kind starpu_worker_get_memory_node_kind (enum starpu_worker_archtype type)

Variables

- struct starpu_worker_collection **starpu_worker_list**
- struct starpu_worker_collection **starpu_worker_tree**

Scheduling operations

- int starpu_worker_sched_op_pending (void)
- void starpu_worker_relax_on (void)
- void starpu_worker_relax_off (void)
- int starpu_worker_get_relax_state (void)
- void starpu_worker_lock (int workerid)
- int starpu_worker_trylock (int workerid)
- void starpu_worker_unlock (int workerid)
- void starpu_worker_lock_self (void)
- void starpu_worker_unlock_self (void)
- void starpu_worker_set_going_to_sleep_callback (void(∗callback)(unsigned workerid))
- void starpu_worker_set_waking_up_callback (void(∗callback)(unsigned workerid))

57.51.1 Detailed Description

57.51.2 Data Structure Documentation

57.51.2.1 struct starpu_sched_ctx_iterator

Structure needed to iterate on the collection

Data Fields

int	cursor	The index of the current worker in the collection, needed when iterating on the collection.
void ∗	value	
void ∗	possible_value	
char	visited[STARPU_NMAXWORKERS]	
int	possibly_parallel	

57.51.2.2 struct starpu_worker_collection

A scheduling context manages a collection of workers that can be memorized using different data structures. Thus, a generic structure is available in order to simplify the choice of its type. Only the list data structure is available but further data structures(like tree) implementations are foreseen.

Data Fields

- int ∗ workerids
- void ∗ **collection_private**
- unsigned nworkers
- void ∗ **unblocked_workers**
- unsigned **nunblocked_workers**
- void ∗ **masters**
- unsigned **nmasters**
- char **present** [STARPU_NMAXWORKERS]
- char **is_unblocked** [STARPU_NMAXWORKERS]
- char **is_master** [STARPU_NMAXWORKERS]
- enum starpu_worker_collection_type type
- unsigned(∗ has_next)(struct starpu_worker_collection ∗workers, struct starpu_sched_ctx_iterator ∗it)
- int(∗ get_next)(struct starpu_worker_collection ∗workers, struct starpu_sched_ctx_iterator ∗it)
- int(∗ add)(struct starpu_worker_collection ∗workers, int worker)
- int(∗ remove)(struct starpu_worker_collection ∗workers, int worker)
- void(∗ init)(struct starpu_worker_collection ∗workers)
- void(∗ deinit)(struct starpu_worker_collection ∗workers)
- void(∗ init_iterator)(struct starpu_worker_collection ∗workers, struct starpu_sched_ctx_iterator ∗it)
- void(∗ **init_iterator_for_parallel_tasks**)(struct starpu_worker_collection ∗workers, struct starpu_sched_ctx_iterator ∗it, struct starpu_task ∗task)

57.51.2.2.1 Field Documentation

57.51.2.2.1.1 workerids `int∗ starpu_worker_collection::workerids`
The workerids managed by the collection

57.51.2.2.1.2 nworkers `unsigned starpu_worker_collection::nworkers`
The number of workers in the collection

57.51.2.2.1.3 type `enum starpu_worker_collection_type starpu_worker_collection::type`
The type of structure

57.51.2.2.1.4 has_next `unsigned(* starpu_worker_collection::has_next) (struct starpu_worker_collection *workers, struct starpu_sched_ctx_iterator *it)`

Check if there is another element in collection

57.51.2.2.1.5 get_next `int(* starpu_worker_collection::get_next) (struct starpu_worker_collection *workers, struct starpu_sched_ctx_iterator *it)`

Return the next element in the collection

57.51.2.2.1.6 add `int(* starpu_worker_collection::add) (struct starpu_worker_collection *workers, int worker)`

Add a new element in the collection

57.51.2.2.1.7 remove `int(* starpu_worker_collection::remove) (struct starpu_worker_collection *workers, int worker)`

Remove an element from the collection

57.51.2.2.1.8 init `void(* starpu_worker_collection::init) (struct starpu_worker_collection *workers)`

Initialize the collection

57.51.2.2.1.9 deinit `void(* starpu_worker_collection::deinit) (struct starpu_worker_collection *workers)`

Deinitialize the collection

57.51.2.2.1.10 init_iterator `void(* starpu_worker_collection::init_iterator) (struct starpu_worker_collection *workers, struct starpu_sched_ctx_iterator *it)`

Initialize the cursor if there is one

57.51.3 Macro Definition Documentation

57.51.3.1 starpu_worker_get_id_check

```
unsigned starpu_worker_get_id_check (
            void )
```

Similar to starpu_worker_get_id(), but abort when called from outside a worker (i.e. when starpu_worker_get_id() would return -1). See How To Initialize A Computation Library Once For Each Worker? for more details.

57.51.3.2 STARPU_MAXNODES

`#define STARPU_MAXNODES`

Define the maximum number of memory nodes managed by StarPU. The default value can be modified at configure by using the option --enable-maxnodes. Reducing it allows to considerably reduce memory used by StarPU data structures.

57.51.3.3 STARPU_MAXCPUS

`#define STARPU_MAXCPUS`

Define the maximum number of CPU workers managed by StarPU. The default value can be modified at configure by using the option --enable-maxcpus.

57.51.3.4 STARPU_MAXNUMANODES

`#define STARPU_MAXNUMANODES`

Define the maximum number of NUMA nodes managed by StarPU. The default value can be modified at configure by using the option --enable-maxnumanodes.

57.51.3.5 STARPU_NMAXWORKERS

`#define STARPU_NMAXWORKERS`

Define the maximum number of workers managed by StarPU.

57.51.3.6 STARPU_UNKNOWN_WORKER

`#define STARPU_UNKNOWN_WORKER`

Invalid worker value

57.51.4 Enumeration Type Documentation

57.51.4.1 starpu_node_kind

`enum starpu_node_kind`

Memory node Type

Enumerator

STARPU_CPU_RAM	CPU core
STARPU_CUDA_RAM	NVIDIA CUDA device
STARPU_OPENCL_RAM	OpenCL device
STARPU_MAX_FPGA_RAM	Maxeler FPGA device
STARPU_DISK_RAM	Disk memory
STARPU_MPI_MS_RAM	MPI Slave device
STARPU_TCPIP_MS_RAM	TCPIP Slave device
STARPU_HIP_RAM	NVIDIA/AMD HIP device
STARPU_MAX_RAM	Maximum value of memory types
STARPU_NRAM	Number of memory types

57.51.4.2 starpu_worker_archtype

`enum starpu_worker_archtype`

Worker Architecture Type

The value 4 which was used by the driver SCC is no longer used as renumbering workers would make unusable old performance model files.

Enumerator

STARPU_CPU_WORKER	CPU core
STARPU_CUDA_WORKER	NVIDIA CUDA device
STARPU_OPENCL_WORKER	OpenCL device
STARPU_MAX_FPGA_WORKER	Maxeler FPGA device
STARPU_MPI_MS_WORKER	MPI Slave device
STARPU_TCPIP_MS_WORKER	TCPIP Slave device
STARPU_HIP_WORKER	NVIDIA/AMD HIP device
STARPU_NARCH	Number of arch types
STARPU_ANY_WORKER	any worker, used in the hypervisor

57.51.4.3 starpu_worker_collection_type

`enum starpu_worker_collection_type`

Types of structures the worker collection can implement

Enumerator

STARPU_WORKER_TREE	The collection is a tree
STARPU_WORKER_LIST	The collection is an array

57.51.5 Function Documentation

57.51.5.1 starpu_worker_wait_for_initialisation()

```
void starpu_worker_wait_for_initialisation (
            void )
```

Wait for all workers to be initialised. Calling this function is normally not necessary. It is called for example in `tools/starpu_machine_display` to make sure all workers information are correctly set before printing their information. See Interleaving StarPU and non-StarPU code for more details.

57.51.5.2 starpu_worker_archtype_is_valid()

```
unsigned starpu_worker_archtype_is_valid (
            enum starpu_worker_archtype type )
```

Return true if type matches one of StarPU's defined worker architectures. See Workers for more details.

57.51.5.3 starpu_arch_mask_to_worker_archtype()

```
enum starpu_worker_archtype starpu_arch_mask_to_worker_archtype (
            unsigned mask )
```

Convert a mask of architectures to a worker archtype. See Workers for more details.

57.51.5.4 starpu_worker_get_count()

```
unsigned starpu_worker_get_count (
            void )
```

Return the number of workers (i.e. processing units executing StarPU tasks). The return value should be at most STARPU_NMAXWORKERS. See Workers for more details.

57.51.5.5 starpu_cpu_worker_get_count()

```
unsigned starpu_cpu_worker_get_count (
            void )
```

Return the number of CPUs controlled by StarPU. The return value should be at most STARPU_MAXCPUS. See Workers for more details.

57.51.5.6 starpu_cuda_worker_get_count()

```
unsigned starpu_cuda_worker_get_count (
            void )
```

Return the number of CUDA devices controlled by StarPU. The return value should be at most STARPU_MAXCUDADEVS. See Workers for more details.

57.51.5.7 starpu_hip_worker_get_count()

```
unsigned starpu_hip_worker_get_count (
            void )
```

Return the number of HIP devices controlled by StarPU. The return value should be at most STARPU_MAXHIPDEVS. See Workers for more details.

57.51.5.8 starpu_opencl_worker_get_count()

```
unsigned starpu_opencl_worker_get_count (
            void )
```

Return the number of OpenCL devices controlled by StarPU. The return value should be at most STARPU_MAXOPENCLDEVS. See Workers for more details.

57.51.5.9 starpu_mpi_ms_worker_get_count()

```
unsigned starpu_mpi_ms_worker_get_count (
            void )
```

Return the number of MPI Master Slave workers controlled by StarPU. See Workers for more details.

57.51.5.10 starpu_tcpip_ms_worker_get_count()

```
unsigned starpu_tcpip_ms_worker_get_count (
            void )
```

Return the number of TCPIP Master Slave workers controlled by StarPU. See Workers for more details.

57.51.5.11 starpu_worker_get_id()

```
int starpu_worker_get_id (
            void )
```

Return the identifier of the current worker, i.e the one associated to the calling thread. The return value is either `-1` if the current context is not a StarPU worker (i.e. when called from the application outside a task or a callback), or an integer between `0` and starpu_worker_get_count() `- 1`. See How To Initialize A Computation Library Once For Each Worker? for more details.

57.51.5.12 starpu_worker_get_bindid()

```
int starpu_worker_get_bindid (
            int workerid )
```

See Workers for more details.

57.51.5.13 starpu_sched_find_all_worker_combinations()

```
void starpu_sched_find_all_worker_combinations (
            void )
```

See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.51.5.14 starpu_worker_get_type()

```
enum starpu_worker_archtype starpu_worker_get_type (
            int id )
```

Return the type of processing unit associated to the worker `id`. The worker identifier is a value returned by the function starpu_worker_get_id()). The return value indicates the architecture of the worker: STARPU_CPU_WORKER for a CPU core, STARPU_CUDA_WORKER for a CUDA device, and STARPU_OPENCL_WORKER for a OpenCL device. The return value for an invalid identifier is unspecified. See Workers for more details.

57.51.5.15 starpu_worker_get_count_by_type()

```
int starpu_worker_get_count_by_type (
            enum starpu_worker_archtype type )
```

Return the number of workers of `type`. A positive (or `NULL`) value is returned in case of success, `-EINVAL` indicates that `type` is not valid otherwise. See Workers for more details.

57.51.5.16 starpu_worker_get_ids_by_type()

```
unsigned starpu_worker_get_ids_by_type (
            enum starpu_worker_archtype type,
            int * workerids,
            unsigned maxsize )
```

Get the list of identifiers of workers of `type`. Fill the array `workerids` with the identifiers of the `workers`. The argument `maxsize` indicates the size of the array `workerids`. The return value gives the number of identifiers that were put in the array. `-ERANGE` is returned is `maxsize` is lower than the number of workers with the appropriate type: in that case, the array is filled with the `maxsize` first elements. To avoid such overflows, the value of maxsize can be chosen by the means of the function starpu_worker_get_count_by_type(), or by passing a value greater or equal to STARPU_NMAXWORKERS. See Workers for more details.

57.51.5.17 starpu_worker_get_by_type()

```
int starpu_worker_get_by_type (
            enum starpu_worker_archtype type,
            int num )
```

Return the identifier of the `num` -th worker that has the specified `type`. If there is no such worker, -1 is returned. See Workers for more details.

57.51.5.18 starpu_worker_get_by_devid()

```
int starpu_worker_get_by_devid (
            enum starpu_worker_archtype type,
            int devid )
```

Return the identifier of the worker that has the specified `type` and device id `devid` (which may not be the n-th, if some devices are skipped for instance). If there is no such worker, `-1` is returned. See Workers for more details.

57.51.5.19 starpu_worker_type_can_execute_task()

```
unsigned starpu_worker_type_can_execute_task (
            enum starpu_worker_archtype worker_type,
            const struct starpu_task * task )
```

Return true if worker type can execute this task. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.51.5.20 starpu_worker_get_name()

```
void starpu_worker_get_name (
            int id,
            char * dst,
            size_t maxlen )
```

Get the name of the worker `id`. StarPU associates a unique human readable string to each processing unit. This function copies at most the `maxlen` first bytes of the unique string associated to the worker `id` into the `dst` buffer. The caller is responsible for ensuring that `dst` is a valid pointer to a buffer of `maxlen` bytes at least. Calling this function on an invalid identifier results in an unspecified behaviour. See Workers for more details.

57.51.5.21 starpu_worker_display_all()

```
void starpu_worker_display_all (
            FILE * output )
```

Display on `output` the list (if any) of all workers. See Workers for more details.

57.51.5.22 starpu_worker_display_names()

```
void starpu_worker_display_names (
             FILE * output,
             enum starpu_worker_archtype type )
```

Display on `output` the list (if any) of all the workers of the given `type`. See Workers for more details.

57.51.5.23 starpu_worker_display_count()

```
void starpu_worker_display_count (
             FILE * output,
             enum starpu_worker_archtype type )
```

Display on `output` the number of workers of the given `type`. See Workers for more details.

57.51.5.24 starpu_worker_get_devid()

```
int starpu_worker_get_devid (
             int id )
```

Return the device id of the worker `id`. The worker should be identified with the value returned by the starpu_worker_get_id() function. In the case of a CUDA worker, this device identifier is the logical device identifier exposed by CUDA (used by the function `cudaGetDevice()` for instance). The device identifier of a CPU worker is the logical identifier of the core on which the worker was bound; this identifier is either provided by the OS or by the library `hwloc` in case it is available. See Workers for more details.

57.51.5.25 starpu_worker_get_devnum()

```
int starpu_worker_get_devnum (
             int id )
```

See Workers for more details.

57.51.5.26 starpu_worker_get_subworkerid()

```
int starpu_worker_get_subworkerid (
             int id )
```

See Workers for more details.

57.51.5.27 starpu_workers_get_tree()

```
struct starpu_tree * starpu_workers_get_tree (
             void  )
```

See Workers for more details.

57.51.5.28 starpu_worker_get_sched_ctx_list()

```
unsigned starpu_worker_get_sched_ctx_list (
             int worker,
             unsigned ** sched_ctx )
```

See Workers for more details.

57.51.5.29 starpu_worker_get_current_task_exp_end()

```
void starpu_worker_get_current_task_exp_end (
             unsigned workerid,
             struct timespec * date )
```

Return when the current task is expected to be finished.

Note: the returned date should be used with caution since the task might very well end just after this function returns.

See Per-task Feedback for more details.

57.51.5.30 starpu_worker_is_blocked_in_parallel()

```
unsigned starpu_worker_is_blocked_in_parallel (
             int workerid )
```

Return whether worker `workerid` is currently blocked in a parallel task. See Helper functions for defining a scheduling policy (Basic
for more details.

57.51.5.31 starpu_worker_is_slave_somewhere()

```
unsigned starpu_worker_is_slave_somewhere (
             int workerid )
```

See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.51.5.32 starpu_worker_get_type_as_string()

```
const char * starpu_worker_get_type_as_string (
             enum starpu_worker_archtype type )
```

Return worker `type` as a string. See Workers for more details.

57.51.5.33 starpu_worker_get_type_from_string()

```
enum starpu_worker_archtype starpu_worker_get_type_from_string (
             const char * type )
```

Return worker `type` from a string. Returns STARPU_UNKNOWN_WORKER if the string doesn't match a worker type. See Workers for more details.

57.51.5.34 starpu_worker_get_type_as_env_var()

```
const char * starpu_worker_get_type_as_env_var (
             enum starpu_worker_archtype type )
```

Return worker `type` as a string suitable for environment variable names (CPU, CUDA, etc.). See Workers for more details.

57.51.5.35 starpu_bindid_get_workerids()

```
int starpu_bindid_get_workerids (
             int bindid,
             int ** workerids )
```

See Workers for more details.

57.51.5.36 starpu_worker_get_devids()

```
int starpu_worker_get_devids (
             enum starpu_worker_archtype type,
             int * devids,
             int num )
```

See Workers for more details.

57.51.5.37 starpu_worker_get_stream_workerids()

```
int starpu_worker_get_stream_workerids (
             unsigned devid,
             int * workerids,
             enum starpu_worker_archtype type )
```

See Workers for more details.

57.51.5.38 starpu_worker_get_hwloc_cpuset()

```
hwloc_cpuset_t starpu_worker_get_hwloc_cpuset (
             int workerid )
```

If StarPU was compiled with `hwloc` support, return a duplicate of the `hwloc` cpuset associated with the worker `workerid`. The returned cpuset is obtained from a `hwloc_bitmap_dup()` function call. It must be freed by the caller using `hwloc_bitmap_free()`. See Interoperability hwloc for more details.

57.51.5.39 starpu_worker_get_hwloc_obj()

```
hwloc_obj_t starpu_worker_get_hwloc_obj (
            int workerid )
```

If StarPU was compiled with `hwloc` support, return the `hwloc` object corresponding to the worker `workerid`. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.51.5.40 starpu_memory_node_get_devid()

```
int starpu_memory_node_get_devid (
            unsigned node )
```

See Memory for more details.

57.51.5.41 starpu_worker_get_local_memory_node()

```
unsigned starpu_worker_get_local_memory_node (
            void )
```

Return the memory node associated to the current worker. See Workers for more details.

57.51.5.42 starpu_worker_get_memory_node()

```
unsigned starpu_worker_get_memory_node (
            unsigned workerid )
```

Return the identifier of the memory node associated to the worker identified by `workerid`. See Workers for more details.

57.51.5.43 starpu_memory_nodes_get_count()

```
unsigned starpu_memory_nodes_get_count (
            void )
```

Return the number of memory nodes. See Workers for more details.

57.51.5.44 starpu_memory_nodes_get_count_by_kind()

```
unsigned starpu_memory_nodes_get_count_by_kind (
            enum starpu_node_kind kind )
```

Return the number of memory nodes of a given `kind`. See Workers for more details.

57.51.5.45 starpu_memory_node_get_ids_by_type()

```
unsigned starpu_memory_node_get_ids_by_type (
            enum starpu_node_kind kind,
            unsigned * memory_nodes_ids,
            unsigned maxsize )
```

Get the list of memory nodes of kind `kind`. Fill the array `memory_nodes_ids` with the memory nodes numbers. The argument `maxsize` indicates the size of the array `memory_nodes_ids`. The return value gives the number of node numbers that were put in the array. `-ERANGE` is returned if `maxsize` is lower than the number of memory nodes with the appropriate kind: in that case, the array is filled with the `maxsize` first elements. To avoid such overflows, the value of maxsize can be chosen by the means of function starpu_memory_nodes_get_count_by_kind(), or by passing a value greater or equal to STARPU_MAXNODES. See Workers for more details.

57.51.5.46 starpu_memory_node_get_name()

```
int starpu_memory_node_get_name (
            unsigned node,
            char * name,
            size_t size )
```

Return in `name` the name of a memory node (NUMA 0, CUDA 0, etc.) `size` is the size of the `name` array. See Workers for more details.

57.51.5.47 starpu_memory_nodes_get_numa_count()

```
unsigned starpu_memory_nodes_get_numa_count (
            void )
```

Return the number of NUMA nodes used by StarPU. See Workers for more details.

57.51.5.48 starpu_memory_nodes_numa_id_to_devid()

```
int starpu_memory_nodes_numa_id_to_devid (
            int osid )
```

Return the identifier of the memory node associated to the NUMA node identified by `osid` by the Operating System. See Workers for more details.

57.51.5.49 starpu_memory_nodes_numa_devid_to_id()

```
int starpu_memory_nodes_numa_devid_to_id (
            unsigned id )
```

Return the Operating System identifier of the memory node whose StarPU identifier is `id`. See Workers for more details.

57.51.5.50 starpu_node_get_kind()

```
enum starpu_node_kind starpu_node_get_kind (
            unsigned node )
```

Return the type of `node` as defined by starpu_node_kind. For example, when defining a new data interface, this function should be used in the allocation function to determine on which device the memory needs to be allocated. See Workers for more details.

57.51.5.51 starpu_memory_node_get_worker_archtype()

```
enum starpu_worker_archtype starpu_memory_node_get_worker_archtype (
            enum starpu_node_kind node_kind )
```

Return the type of worker which operates on memory node kind `node_kind`. See Workers for more details.

57.51.5.52 starpu_worker_get_memory_node_kind()

```
enum starpu_node_kind starpu_worker_get_memory_node_kind (
            enum starpu_worker_archtype type )
```

Return the type of memory node that arch type `type` operates on. See Workers for more details.

57.51.5.53 starpu_worker_sched_op_pending()

```
int starpu_worker_sched_op_pending (
            void )
```

Return `!0` if current worker has a scheduling operation in progress, and `0` otherwise.

57.51.5.54 starpu_worker_relax_on()

```
void starpu_worker_relax_on (
            void )
```

Allow other threads and workers to temporarily observe the current worker state, even though it is performing a scheduling operation. Must be called by a worker before performing a potentially blocking call such as acquiring a mutex other than its own sched_mutex. This function increases `state_relax_refcnt` from the current worker. No more than `UINT_MAX-1` nested starpu_worker_relax_on() calls should performed on the same worker. This function is automatically called by starpu_worker_lock() to relax the caller worker state while attempting to lock the target worker. See Defining A New Basic Scheduling Policy for more details.

57.51.5.55 starpu_worker_relax_off()

```
void starpu_worker_relax_off (
            void )
```

Must be called after a potentially blocking call is complete, to restore the relax state in place before the corresponding starpu_worker_relax_on(). Decreases `state_relax_refcnt`. Calls to starpu_worker_relax_on() and starpu_worker_relax_off() must be properly paired. This function is automatically called by starpu_worker_unlock() after the target worker has been unlocked. See Defining A New Basic Scheduling Policy for more details.

57.51.5.56 starpu_worker_get_relax_state()

```
int starpu_worker_get_relax_state (
            void )
```

Return `!0` if the current worker `state_relax_refcnt!=0` and `0` otherwise. See Defining A New Basic Scheduling Policy for more details.

57.51.5.57 starpu_worker_lock()

```
void starpu_worker_lock (
            int workerid )
```

Acquire the sched mutex of `workerid`. If the caller is a worker, distinct from `workerid`, the caller worker automatically enters a relax state while acquiring the target worker lock. See Defining A New Basic Scheduling Policy for more details.

57.51.5.58 starpu_worker_trylock()

```
int starpu_worker_trylock (
            int workerid )
```

Attempt to acquire the sched mutex of `workerid`. Returns `0` if successful, `!0` if `workerid` sched mutex is held or the corresponding worker is not in a relax state. If the caller is a worker, distinct from `workerid`, the caller worker automatically enters relax state if successfully acquiring the target worker lock. See Defining A New Basic Scheduling Policy for more details.

57.51.5.59 starpu_worker_unlock()

```
void starpu_worker_unlock (
            int workerid )
```

Release the previously acquired sched mutex of `workerid`. Restore the relax state of the caller worker if needed. See Defining A New Basic Scheduling Policy for more details.

57.51.5.60 starpu_worker_lock_self()

```
void starpu_worker_lock_self (
            void )
```

Acquire the current worker sched mutex. See Defining A New Basic Scheduling Policy for more details.

57.51.5.61 starpu_worker_unlock_self()

```
void starpu_worker_unlock_self (
            void )
```

Release the current worker sched mutex. See Defining A New Basic Scheduling Policy for more details.

57.51.5.62 starpu_worker_set_going_to_sleep_callback()

```
void starpu_worker_set_going_to_sleep_callback (
            void(*)(unsigned workerid) callback )
```

If StarPU was compiled with blocking drivers support and worker callbacks support enabled, allow to specify an external resource manager callback to be notified about workers going to sleep. See Helper functions for defining a scheduling policy (Basic or modular) for more details.

57.51.5.63 starpu_worker_set_waking_up_callback()

```
void starpu_worker_set_waking_up_callback (
            void(*)(unsigned workerid) callback )
```

If StarPU was compiled with blocking drivers support and worker callbacks support enabled, allow to specify an external resource manager callback to be notified about workers waking-up. See Helper functions for defining a scheduling policy (Basic
for more details.

Chapter 58

File Index

58.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 59

File Documentation

59.1 starpu.h File Reference

```
#include <stdlib.h>
#include <stdint.h>
#include <starpu_config.h>
#include <windows.h>
#include <starpu_opencl.h>
#include <starpu_thread.h>
#include <starpu_thread_util.h>
#include <starpu_util.h>
#include <starpu_data.h>
#include <starpu_helper.h>
#include <starpu_disk.h>
#include <starpu_data_interfaces.h>
#include <starpu_data_filters.h>
#include <starpu_stdlib.h>
#include <starpu_task_bundle.h>
#include <starpu_task_dep.h>
#include <starpu_task.h>
#include <starpu_worker.h>
#include <starpu_perfmodel.h>
#include <starpu_task_list.h>
#include <starpu_task_util.h>
#include <starpu_scheduler.h>
#include <starpu_sched_ctx.h>
#include <starpu_expert.h>
#include <starpu_rand.h>
#include <starpu_cuda.h>
#include <starpu_hip.h>
#include <starpu_hipblas.h>
#include <starpu_cublas.h>
#include <starpu_cusparse.h>
#include <starpu_bound.h>
#include <starpu_hash.h>
#include <starpu_profiling.h>
#include <starpu_profiling_tool.h>
#include <starpu_fxt.h>
#include <starpu_driver.h>
#include <starpu_tree.h>
#include <starpu_openmp.h>
#include <starpu_simgrid_wrap.h>
#include <starpu_bitmap.h>
#include <starpu_parallel_worker.h>
```

```
#include <starpu_perf_monitoring.h>
#include <starpu_perf_steering.h>
#include <starpu_max_fpga.h>
#include "starpu_deprecated_api.h"
```

Data Structures

- struct starpu_conf

Macros

- #define STARPU_THREAD_ACTIVE

Functions

- int starpu_conf_init (struct starpu_conf ∗conf)
- int starpu_conf_noworker (struct starpu_conf ∗conf)
- int starpu_init (struct starpu_conf ∗conf)
- int starpu_initialize (struct starpu_conf ∗user_conf, int ∗argc, char ∗∗∗argv)
- int starpu_is_initialized (void)
- void starpu_wait_initialized (void)
- void starpu_shutdown (void)
- void starpu_pause (void)
- void starpu_resume (void)
- int starpu_is_paused (void)
- unsigned starpu_get_next_bindid (unsigned flags, unsigned ∗preferred, unsigned npreferred)
- int starpu_bind_thread_on (int cpuid, unsigned flags, const char ∗name)
- void starpu_bind_thread_on_worker (unsigned workerid)
- void starpu_bind_thread_on_main (void)
- void starpu_bind_thread_on_cpu (int cpuid)
- int starpu_cpu_os_index (int cpuid)
- void starpu_topology_print (FILE ∗f)
- int starpu_asynchronous_copy_disabled (void)
- int starpu_asynchronous_cuda_copy_disabled (void)
- int starpu_asynchronous_hip_copy_disabled (void)
- int starpu_asynchronous_opencl_copy_disabled (void)
- int starpu_asynchronous_max_fpga_copy_disabled (void)
- int starpu_asynchronous_mpi_ms_copy_disabled (void)
- int starpu_asynchronous_tcpip_ms_copy_disabled (void)
- int starpu_asynchronous_copy_disabled_for (enum starpu_node_kind kind)
- int starpu_map_enabled (void)
- void starpu_display_stats (void)
- void starpu_get_version (int ∗major, int ∗minor, int ∗release)

59.2 starpu_bitmap.h File Reference

```
#include <starpu_util.h>
#include <starpu_config.h>
#include <string.h>
#include <stdlib.h>
```

Data Structures

- struct starpu_bitmap

Macros

- #define **_STARPU_LONG_BIT**
- #define **_STARPU_BITMAP_SIZE**
- #define **_starpu_check_bitmap**(b)

Functions

- static struct starpu_bitmap ∗ starpu_bitmap_create (void) STARPU_ATTRIBUTE_MALLOC
- static void starpu_bitmap_init (struct starpu_bitmap ∗b)
- static void starpu_bitmap_destroy (struct starpu_bitmap ∗b)
- static void starpu_bitmap_set (struct starpu_bitmap ∗b, int e)
- static void starpu_bitmap_unset (struct starpu_bitmap ∗b, int e)
- static void starpu_bitmap_unset_all (struct starpu_bitmap ∗b)
- static int starpu_bitmap_get (struct starpu_bitmap ∗b, int e)
- static void starpu_bitmap_unset_and (struct starpu_bitmap ∗a, struct starpu_bitmap ∗b, struct starpu_bitmap ∗c)
- static void starpu_bitmap_or (struct starpu_bitmap ∗a, struct starpu_bitmap ∗b)
- static int starpu_bitmap_and_get (struct starpu_bitmap ∗b1, struct starpu_bitmap ∗b2, int e)
- static int starpu_bitmap_cardinal (struct starpu_bitmap ∗b)
- static int starpu_bitmap_first (struct starpu_bitmap ∗b)
- static int starpu_bitmap_last (struct starpu_bitmap ∗b)
- static int starpu_bitmap_next (struct starpu_bitmap ∗b, int e)
- static int starpu_bitmap_has_next (struct starpu_bitmap ∗b, int e)
- static int **_starpu_count_bit_static** (unsigned long e)
- static int **_starpu_get_first_bit_rank** (unsigned long ms)
- static int **_starpu_get_last_bit_rank** (unsigned long l)

59.2.1 Data Structure Documentation

59.2.1.1 struct starpu_bitmap

todo

Data Fields

unsigned long	bits[_STARPU_BITMAP_SIZE]	
int	cardinal	

59.3 starpu_bound.h File Reference

`#include <stdio.h>`

Functions

- void starpu_bound_start (int deps, int prio)
- void starpu_bound_stop (void)
- void starpu_bound_print_dot (FILE ∗output)
- void starpu_bound_compute (double ∗res, double ∗integer_res, int integer)
- void starpu_bound_print_lp (FILE ∗output)
- void starpu_bound_print_mps (FILE ∗output)
- void starpu_bound_print (FILE ∗output, int integer)

59.4 starpu_config.h File Reference

`#include <sys/types.h>`

Macros

- #define STARPU_MAJOR_VERSION
- #define STARPU_MINOR_VERSION
- #define STARPU_RELEASE_VERSION
- #define **STARPU_USE_CPU**
- #define STARPU_USE_CUDA
- #define STARPU_USE_CUDA0
- #define STARPU_USE_CUDA1
- #define STARPU_USE_HIP
- #define STARPU_HAVE_NVML_H
- #define STARPU_USE_OPENCL
- #define STARPU_USE_MAX_FPGA
- #define STARPU_USE_MPI_MASTER_SLAVE
- #define STARPU_USE_TCPIP_MASTER_SLAVE
- #define STARPU_OPENMP
- #define **STARPU_BUBBLE**
- #define **STARPU_PARALLEL_WORKER**
- #define **STARPU_SIMGRID**
- #define **STARPU_SIMGRID_MC**
- #define **STARPU_SIMGRID_HAVE_XBT_BARRIER_INIT**
- #define **STARPU_HAVE_SIMGRID_MSG_H**
- #define **STARPU_HAVE_MSG_MSG_H**
- #define **STARPU_HAVE_SIMGRID_ACTOR_H**
- #define **STARPU_HAVE_SIMGRID_SEMAPHORE_H**
- #define **STARPU_HAVE_SIMGRID_MUTEX_H**
- #define **STARPU_HAVE_SIMGRID_COND_H**
- #define **STARPU_HAVE_SIMGRID_BARRIER_H**
- #define **STARPU_HAVE_XBT_SYNCHRO_H**
- #define **STARPU_HAVE_VALGRIND_H**
- #define **STARPU_HAVE_MEMCHECK_H**
- #define **STARPU_VALGRIND_FULL**
- #define **STARPU_SANITIZE_LEAK**
- #define **STARPU_NON_BLOCKING_DRIVERS**
- #define **STARPU_WORKER_CALLBACKS**
- #define **STARPU_HAVE_ICC**
- #define STARPU_USE_MPI
- #define **STARPU_USE_MPI_MPI**
- #define **STARPU_USE_MPI_NMAD**
- #define **STARPU_USE_MPI_FT**
- #define **STARPU_USE_MPI_FT_STATS**
- #define **STARPU_ATLAS**
- #define **STARPU_GOTO**
- #define **STARPU_OPENBLAS**
- #define **STARPU_MKL**
- #define **STARPU_ARMPL**
- #define **STARPU_SYSTEM_BLAS**
- #define **STARPU_HAVE_CBLAS_H**
- #define **STARPU_HAVE_BLAS**
- #define STARPU_OPENCL_DATADIR

- #define **STARPU_HAVE_LIBCUSPARSE**
- #define **STARPU_HAVE_LIBCUSOLVER**
- #define **STARPU_HAVE_MAGMA**
- #define **STARPU_OPENGL_RENDER**
- #define **STARPU_USE_GTK**
- #define **STARPU_HAVE_X11**
- #define **STARPU_PAPI**
- #define **STARPU_HAVE_POSIX_MEMALIGN**
- #define **STARPU_HAVE_MEMALIGN**
- #define **STARPU_HAVE_MALLOC_H**
- #define **STARPU_HAVE_SYNC_BOOL_COMPARE_AND_SWAP**
- #define **STARPU_HAVE_SYNC_BOOL_COMPARE_AND_SWAP_8**
- #define **STARPU_HAVE_SYNC_VAL_COMPARE_AND_SWAP**
- #define **STARPU_HAVE_SYNC_VAL_COMPARE_AND_SWAP_8**
- #define **STARPU_HAVE_SYNC_FETCH_AND_ADD**
- #define **STARPU_HAVE_SYNC_FETCH_AND_ADD_8**
- #define **STARPU_HAVE_SYNC_FETCH_AND_OR**
- #define **STARPU_HAVE_SYNC_FETCH_AND_OR_8**
- #define **STARPU_HAVE_SYNC_LOCK_TEST_AND_SET**
- #define **STARPU_HAVE_ATOMIC_COMPARE_EXCHANGE_N**
- #define **STARPU_HAVE_ATOMIC_COMPARE_EXCHANGE_N_8**
- #define **STARPU_HAVE_ATOMIC_EXCHANGE_N**
- #define **STARPU_HAVE_ATOMIC_EXCHANGE_N_8**
- #define **STARPU_HAVE_ATOMIC_FETCH_ADD**
- #define **STARPU_HAVE_ATOMIC_FETCH_ADD_8**
- #define **STARPU_HAVE_ATOMIC_FETCH_OR**
- #define **STARPU_HAVE_ATOMIC_FETCH_OR_8**
- #define **STARPU_HAVE_ATOMIC_TEST_AND_SET**
- #define **STARPU_HAVE_SYNC_SYNCHRONIZE**
- #define **STARPU_DEVEL**
- #define **STARPU_MODEL_DEBUG**
- #define **STARPU_NO_ASSERT**
- #define **STARPU_DEBUG**
- #define **STARPU_VERBOSE**
- #define **STARPU_GDB_PATH**
- #define **STARPU_HAVE_FFTW**
- #define **STARPU_HAVE_FFTWF**
- #define **STARPU_HAVE_FFTWL**
- #define **STARPU_HAVE_CUFFTDOUBLECOMPLEX**
- #define **STARPU_HAVE_CURAND**
- #define STARPU_MAXNODES
- #define STARPU_NMAXBUFS
- #define STARPU_FXT_MAX_FILES
- #define STARPU_MAXCPUS
- #define STARPU_MAXNUMANODES
- #define STARPU_MAXCUDADEVS
- #define STARPU_MAXOPENCLDEVS
- #define STARPU_MAXMAXFPGADEVS
- #define STARPU_MAXHIPDEVS
- #define STARPU_NMAXWORKERS
- #define STARPU_NMAX_SCHED_CTXS
- #define STARPU_MAXIMPLEMENTATIONS
- #define **STARPU_USE_SC_HYPERVISOR**
- #define **STARPU_SC_HYPERVISOR_DEBUG**
- #define **STARPU_HAVE_GLPK_H**

- #define **STARPU_HAVE_CUDA_MEMCPY_PEER**
- #define **STARPU_HAVE_LIBNUMA**
- #define **STARPU_HAVE_WINDOWS**
- #define **STARPU_LINUX_SYS**
- #define **STARPU_HAVE_SETENV**
- #define **STARPU_HAVE_UNSETENV**
- #define **STARPU_HAVE_UNISTD_H**
- #define **STARPU_HAVE_HDF5**
- #define **STARPU_HAVE_MPI_COMM_CREATE_GROUP**
- #define **STARPU_USE_FXT**
- #define **STARPU_FXT_LOCK_TRACES**
- #define **__starpu_func__**
- #define **__starpu_inline**
- #define **STARPU_QUICK_CHECK**
- #define **STARPU_LONG_CHECK**
- #define **STARPU_USE_DRAND48**
- #define **STARPU_USE_ERAND48_R**
- #define **STARPU_HAVE_NEARBYINTF**
- #define **STARPU_HAVE_RINTF**
- #define **STARPU_HAVE_HWLOC**
- #define **STARPU_HAVE_PTHREAD_SPIN_LOCK**
- #define **STARPU_HAVE_PTHREAD_BARRIER**
- #define **STARPU_HAVE_PTHREAD_SETNAME_NP**
- #define **STARPU_HAVE_STRUCT_TIMESPEC**
- #define **STARPU_PTHREAD_MUTEX_INITIALIZER_ZERO**
- #define **STARPU_PTHREAD_COND_INITIALIZER_ZERO**
- #define **STARPU_PTHREAD_RWLOCK_INITIALIZER_ZERO**
- #define STARPU_HAVE_HELGRIND_H
- #define HAVE_MPI_COMM_F2C
- #define **STARPU_HAVE_DARWIN**
- #define **STARPU_HAVE_CXX11**
- #define **STARPU_HAVE_STRERROR_R**
- #define **STARPU_HAVE_STATEMENT_EXPRESSIONS**
- #define **STARPU_PERF_MODEL_DIR**
- #define **STARPU_PYTHON_HAVE_NUMPY**
- #define **STARPU_PROF_TOOL**

Typedefs

- typedef ssize_t **starpu_ssize_t**

59.4.1 Macro Definition Documentation

59.4.1.1 STARPU_USE_CUDA0

`#define STARPU_USE_CUDA0`

Defined when StarPU is testing the CUDA0 driver.

59.4.1.2 STARPU_USE_CUDA1

`#define STARPU_USE_CUDA1`

Defined when StarPU is testing the CUDA1 driver.

59.4.1.3 STARPU_USE_TCPIP_MASTER_SLAVE

`#define STARPU_USE_TCPIP_MASTER_SLAVE`

Defined when StarPU has been installed with TCP/IP Master Slave support. It should be used in your code to detect the availability of TCP/IP Master Slave.

59.4.1.4 STARPU_HAVE_HELGRIND_H

`#define STARPU_HAVE_HELGRIND_H`

This is only for building examples

59.4.1.5 HAVE_MPI_COMM_F2C

`#define HAVE_MPI_COMM_F2C`

Enable Fortran to C MPI interface

59.5 starpu_cublas.h File Reference

Functions

- void starpu_cublas_init (void)
- void starpu_cublas_set_stream (void)
- void starpu_cublas_shutdown (void)

59.6 starpu_cublas_v2.h File Reference

```
#include <cublas_v2.h>
```

Functions

- cublasHandle_t starpu_cublas_get_local_handle (void)

59.7 starpu_cusparse.h File Reference

```
#include <cusparse.h>
```

Functions

- void starpu_cusparse_init (void)
- void starpu_cusparse_shutdown (void)
- cusparseHandle_t starpu_cusparse_get_local_handle (void)

59.8 starpu_cuda.h File Reference

```
#include <starpu_config.h>
#include <cuda.h>
#include <cuda_runtime.h>
#include <cuda_runtime_api.h>
#include <nvml.h>
```

Macros

- #define STARPU_CUBLAS_REPORT_ERROR(status)
- #define STARPU_CUDA_REPORT_ERROR(status)

Functions

- void starpu_cublas_report_error (const char ∗func, const char ∗file, int line, int status)
- void starpu_cuda_report_error (const char ∗func, const char ∗file, int line, cudaError_t status)
- cudaStream_t starpu_cuda_get_local_stream (void)
- const struct cudaDeviceProp ∗ starpu_cuda_get_device_properties (unsigned workerid)
- int starpu_cuda_copy_async_sync (void ∗src_ptr, unsigned src_node, void ∗dst_ptr, unsigned dst_node, size_t ssize, cudaStream_t stream, enum cudaMemcpyKind kind)
- int starpu_cuda_copy2d_async_sync (void ∗src_ptr, unsigned src_node, void ∗dst_ptr, unsigned dst_node, size_t blocksize, size_t numblocks, size_t ld_src, size_t ld_dst, cudaStream_t stream, enum cudaMemcpy↩ Kind kind)
- int starpu_cuda_copy3d_async_sync (void ∗src_ptr, unsigned src_node, void ∗dst_ptr, unsigned dst_node, size_t blocksize, size_t numblocks_1, size_t ld1_src, size_t ld1_dst, size_t numblocks_2, size_t ld2_src, size_t ld2_dst, cudaStream_t stream, enum cudaMemcpyKind kind)
- void starpu_cuda_set_device (unsigned devid)
- nvmlDevice_t starpu_cuda_get_nvmldev (unsigned devid)

59.9 starpu_data.h File Reference

`#include <starpu.h>`

Typedefs

- typedef struct _starpu_data_state ∗ starpu_data_handle_t
- typedef struct starpu_arbiter ∗ starpu_arbiter_t

Enumerations

- enum starpu_data_access_mode {
  STARPU_NONE , STARPU_R , STARPU_W , STARPU_RW ,
  STARPU_SCRATCH , STARPU_REDUX , STARPU_COMMUTE , STARPU_SSEND ,
  STARPU_LOCALITY , STARPU_MPI_REDUX , STARPU_NOPLAN , STARPU_UNMAP ,
  STARPU_NOFOOTPRINT , STARPU_ACCESS_MODE_MAX }
- enum starpu_is_prefetch {
  STARPU_FETCH , STARPU_TASK_PREFETCH , STARPU_PREFETCH , STARPU_IDLEFETCH ,
  STARPU_NFETCH }

Functions

- void starpu_data_set_name (starpu_data_handle_t handle, const char ∗name)
- void starpu_data_set_coordinates_array (starpu_data_handle_t handle, unsigned dimensions, int dims[ ])
- void starpu_data_set_coordinates (starpu_data_handle_t handle, unsigned dimensions,...)
- unsigned starpu_data_get_coordinates_array (starpu_data_handle_t handle, unsigned dimensions, int dims[ ])
- void starpu_data_unregister (starpu_data_handle_t handle)
- void starpu_data_unregister_no_coherency (starpu_data_handle_t handle)
- void starpu_data_unregister_submit (starpu_data_handle_t handle)
- void starpu_data_deinitialize (starpu_data_handle_t handle)
- void starpu_data_deinitialize_submit (starpu_data_handle_t handle)
- void starpu_data_invalidate (starpu_data_handle_t handle)

- void starpu_data_invalidate_submit (starpu_data_handle_t handle)
- void starpu_data_advise_as_important (starpu_data_handle_t handle, unsigned is_important)
- starpu_arbiter_t starpu_arbiter_create (void) STARPU_ATTRIBUTE_MALLOC
- void starpu_data_assign_arbiter (starpu_data_handle_t handle, starpu_arbiter_t arbiter)
- void starpu_arbiter_destroy (starpu_arbiter_t arbiter)
- int starpu_data_request_allocation (starpu_data_handle_t handle, unsigned node)
- int starpu_data_fetch_on_node (starpu_data_handle_t handle, unsigned node, unsigned async)
- int starpu_data_prefetch_on_node (starpu_data_handle_t handle, unsigned node, unsigned async)
- int starpu_data_prefetch_on_node_prio (starpu_data_handle_t handle, unsigned node, unsigned async, int prio)
- int starpu_data_idle_prefetch_on_node (starpu_data_handle_t handle, unsigned node, unsigned async)
- int starpu_data_idle_prefetch_on_node_prio (starpu_data_handle_t handle, unsigned node, unsigned async, int prio)
- unsigned starpu_data_is_on_node (starpu_data_handle_t handle, unsigned node)
- void starpu_data_wont_use (starpu_data_handle_t handle)
- int starpu_data_evict_from_node (starpu_data_handle_t handle, unsigned node)
- void starpu_data_set_wt_mask (starpu_data_handle_t handle, uint32_t wt_mask)
- void starpu_data_set_ooc_flag (starpu_data_handle_t handle, unsigned flag)
- unsigned starpu_data_get_ooc_flag (starpu_data_handle_t handle)
- void starpu_data_query_status2 (starpu_data_handle_t handle, int memory_node, int ∗is_allocated, int ∗is_valid, int ∗is_loading, int ∗is_requested)
- void starpu_data_query_status (starpu_data_handle_t handle, int memory_node, int ∗is_allocated, int ∗is_valid, int ∗is_requested)
- void starpu_data_set_reduction_methods (starpu_data_handle_t handle, struct starpu_codelet ∗redux_cl, struct starpu_codelet ∗init_cl)
- void starpu_data_set_reduction_methods_with_args (starpu_data_handle_t handle, struct starpu_codelet ∗redux_cl, void ∗redux_cl_arg, struct starpu_codelet ∗init_cl, void ∗init_cl_arg)
- struct starpu_data_interface_ops ∗ **starpu_data_get_interface_ops** (starpu_data_handle_t handle)
- unsigned starpu_data_test_if_allocated_on_node (starpu_data_handle_t handle, unsigned memory_node)
- unsigned starpu_data_test_if_mapped_on_node (starpu_data_handle_t handle, unsigned memory_node)
- void starpu_memchunk_tidy (unsigned memory_node)
- void starpu_data_set_user_data (starpu_data_handle_t handle, void ∗user_data)
- void ∗ starpu_data_get_user_data (starpu_data_handle_t handle)
- void starpu_data_set_sched_data (starpu_data_handle_t handle, void ∗sched_data)
- void ∗ starpu_data_get_sched_data (starpu_data_handle_t handle)
- int starpu_data_can_evict (starpu_data_handle_t handle, unsigned node, enum starpu_is_prefetch is_prefetch)

Implicit Data Dependencies

In this section, we describe how StarPU makes it possible to insert implicit task dependencies in order to enforce sequential data consistency. When this data consistency is enabled on a specific data handle, any data access will appear as sequentially consistent from the application. For instance, if the application submits two tasks that access the same piece of data in read-only mode, and then a third task that access it in write mode, dependencies will be added between the two first tasks and the third one. Implicit data dependencies are also inserted in the case of data accesses from the application.

- void starpu_data_set_sequential_consistency_flag (starpu_data_handle_t handle, unsigned flag)
- unsigned starpu_data_get_sequential_consistency_flag (starpu_data_handle_t handle)
- unsigned starpu_data_get_default_sequential_consistency_flag (void)
- void starpu_data_set_default_sequential_consistency_flag (unsigned flag)

Access registered data from the application

- #define STARPU_ACQUIRE_NO_NODE
- #define STARPU_ACQUIRE_NO_NODE_LOCK_ALL
- #define STARPU_DATA_ACQUIRE_CB(handle, mode, code)
- int starpu_data_acquire (starpu_data_handle_t handle, enum starpu_data_access_mode mode)
- int starpu_data_acquire_on_node (starpu_data_handle_t handle, int node, enum starpu_data_access_mode mode)
- int starpu_data_acquire_cb (starpu_data_handle_t handle, enum starpu_data_access_mode mode, void(∗callback)(void ∗), void ∗arg)
- int starpu_data_acquire_on_node_cb (starpu_data_handle_t handle, int node, enum starpu_data_access_mode mode, void(∗callback)(void ∗), void ∗arg)
- int starpu_data_acquire_cb_sequential_consistency (starpu_data_handle_t handle, enum starpu_data_access_mode mode, void(∗callback)(void ∗), void ∗arg, int sequential_consistency)
- int starpu_data_acquire_on_node_cb_sequential_consistency (starpu_data_handle_t handle, int node, enum starpu_data_access_mode mode, void(∗callback)(void ∗), void ∗arg, int sequential_consistency)
- int starpu_data_acquire_on_node_cb_sequential_consistency_sync_jobids (starpu_data_handle_t handle, int node, enum starpu_data_access_mode mode, void(∗callback_acquired)(void ∗arg, int ∗node, enum starpu_data_access_mode mode), void(∗callback)(void ∗arg), void ∗arg, int sequential_consistency, int quick, long ∗pre_sync_jobid, long ∗post_sync_jobid, int prio)
- int starpu_data_acquire_try (starpu_data_handle_t handle, enum starpu_data_access_mode mode)
- int starpu_data_acquire_on_node_try (starpu_data_handle_t handle, int node, enum starpu_data_access_mode mode)
- void starpu_data_release (starpu_data_handle_t handle)
- void starpu_data_release_on_node (starpu_data_handle_t handle, int node)
- void starpu_data_release_to (starpu_data_handle_t handle, enum starpu_data_access_mode down_to_mode)
- void starpu_data_release_to_on_node (starpu_data_handle_t handle, enum starpu_data_access_mode down_to_mode, int node)

59.10 starpu_data_filters.h File Reference

```
#include <starpu.h>
#include <stdarg.h>
```

Data Structures

- struct starpu_data_filter

Functions

Basic API

- void starpu_data_partition (starpu_data_handle_t initial_handle, struct starpu_data_filter ∗f)
- void starpu_data_unpartition (starpu_data_handle_t root_data, unsigned gathering_node)
- starpu_data_handle_t starpu_data_get_child (starpu_data_handle_t handle, unsigned i)
- int starpu_data_get_nb_children (starpu_data_handle_t handle)
- starpu_data_handle_t starpu_data_get_sub_data (starpu_data_handle_t root_data, unsigned depth,...)
- starpu_data_handle_t starpu_data_vget_sub_data (starpu_data_handle_t root_data, unsigned depth, va_list pa)
- void starpu_data_map_filters (starpu_data_handle_t root_data, unsigned nfilters,...)
- void starpu_data_vmap_filters (starpu_data_handle_t root_data, unsigned nfilters, va_list pa)
- void starpu_data_map_filters_parray (starpu_data_handle_t root_handle, int nfilters, struct starpu_data_filter ∗∗filters)
- void starpu_data_map_filters_array (starpu_data_handle_t root_handle, int nfilters, struct starpu_data_filter ∗filters)

Asynchronous API

- void starpu_data_partition_plan (starpu_data_handle_t initial_handle, struct starpu_data_filter ∗f, starpu_data_handle_t ∗children)
- void starpu_data_partition_submit (starpu_data_handle_t initial_handle, unsigned nparts, starpu_data_handle_t ∗children)
- void starpu_data_partition_readonly_submit (starpu_data_handle_t initial_handle, unsigned nparts, starpu_data_handle_t ∗children)
- void starpu_data_partition_readonly_submit_sequential_consistency (starpu_data_handle_t initial_↩ handle, unsigned nparts, starpu_data_handle_t ∗children, int sequential_consistency)
- void starpu_data_partition_readwrite_upgrade_submit (starpu_data_handle_t initial_handle, unsigned nparts, starpu_data_handle_t ∗children)
- void starpu_data_partition_readonly_downgrade_submit (starpu_data_handle_t initial_handle, unsigned nparts, starpu_data_handle_t ∗children)
- void starpu_data_unpartition_submit (starpu_data_handle_t initial_handle, unsigned nparts, starpu_data_handle_t ∗children, int gathering_node)
- void starpu_data_unpartition_readonly_submit (starpu_data_handle_t initial_handle, unsigned nparts, starpu_data_handle_t ∗children, int gathering_node)
- void starpu_data_partition_clean (starpu_data_handle_t root_data, unsigned nparts, starpu_data_handle_t ∗children)
- void starpu_data_partition_clean_node (starpu_data_handle_t root_data, unsigned nparts, starpu_data_handle_t ∗children, int gather_node)
- void starpu_data_unpartition_submit_sequential_consistency_cb (starpu_data_handle_t initial_↩ handle, unsigned nparts, starpu_data_handle_t ∗children, int gather_node, int sequential_consistency, void(∗callback_func)(void ∗), void ∗callback_arg)
- void starpu_data_partition_submit_sequential_consistency (starpu_data_handle_t initial_handle, unsigned nparts, starpu_data_handle_t ∗children, int sequential_consistency)
- void starpu_data_unpartition_submit_sequential_consistency (starpu_data_handle_t initial_handle, unsigned nparts, starpu_data_handle_t ∗children, int gathering_node, int sequential_consistency)

Predefined BCSR Filter Functions

Predefined partitioning functions for BCSR data. Examples on how to use them are shown in Partitioning Data.

- void starpu_bcsr_filter_canonical_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- unsigned starpu_bcsr_filter_canonical_block_get_nchildren (struct starpu_data_filter ∗f, starpu_data_handle_t handle)
- struct starpu_data_interface_ops ∗ starpu_bcsr_filter_canonical_block_child_ops (struct starpu_data_filter ∗f, unsigned child)
- void starpu_bcsr_filter_vertical_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)

Predefined CSR Filter Functions

Predefined partitioning functions for CSR data. Examples on how to use them are shown in Partitioning Data.

- void starpu_csr_filter_vertical_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)

Predefined Matrix Filter Functions

Predefined partitioning functions for matrix data. Examples on how to use them are shown in Partitioning Data. Note: this is using the C element order which is row-major, i.e. elements with consecutive x coordinates are consecutive in memory.

- void starpu_matrix_filter_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_matrix_filter_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_matrix_filter_vertical_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_matrix_filter_vertical_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)

- void starpu_matrix_filter_pick_vector_y (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- struct starpu_data_interface_ops ∗ starpu_matrix_filter_pick_vector_child_ops (struct starpu_data_filter ∗f, unsigned child)
- void starpu_matrix_filter_pick_variable (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- struct starpu_data_interface_ops ∗ starpu_matrix_filter_pick_variable_child_ops (struct starpu_data_filter ∗f, unsigned child)

Predefined Vector Filter Functions

Predefined partitioning functions for vector data. Examples on how to use them are shown in Partitioning Data.

- void starpu_vector_filter_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_vector_filter_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_vector_filter_list_long (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_vector_filter_list (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_vector_filter_divide_in_2 (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_vector_filter_pick_variable (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- struct starpu_data_interface_ops ∗ starpu_vector_filter_pick_variable_child_ops (struct starpu_data_filter ∗f, unsigned child)

Predefined Block Filter Functions

Predefined partitioning functions for block data. Examples on how to use them are shown in Partitioning Data. An example is available in `examples/filters/shadow3d.c` Note: this is using the C element order which is row-major, i.e. elements with consecutive x coordinates are consecutive in memory.

- void starpu_block_filter_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_block_filter_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_block_filter_vertical_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_block_filter_vertical_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_block_filter_depth_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_block_filter_depth_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_block_filter_pick_matrix_z (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_block_filter_pick_matrix_y (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- struct starpu_data_interface_ops ∗ starpu_block_filter_pick_matrix_child_ops (struct starpu_data_filter ∗f, unsigned child)
- void starpu_block_filter_pick_variable (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- struct starpu_data_interface_ops ∗ starpu_block_filter_pick_variable_child_ops (struct starpu_data_filter ∗f, unsigned child)

Predefined Tensor Filter Functions

Predefined partitioning functions for tensor data.

- void starpu_tensor_filter_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)

- void starpu_tensor_filter_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_vertical_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_vertical_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_depth_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_depth_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_time_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_time_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_pick_block_t (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_pick_block_z (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_tensor_filter_pick_block_y (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- struct starpu_data_interface_ops ∗ starpu_tensor_filter_pick_block_child_ops (struct starpu_data_filter ∗f, unsigned child)
- void starpu_tensor_filter_pick_variable (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- struct starpu_data_interface_ops ∗ starpu_tensor_filter_pick_variable_child_ops (struct starpu_data_filter ∗f, unsigned child)

Predefined Ndim Filter Functions

Predefined partitioning functions for ndim array data.

- void starpu_ndim_filter_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_block_shadow (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_to_tensor (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_to_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_to_matrix (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_to_vector (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_to_variable (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_pick_ndim (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_5d_pick_tensor (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_4d_pick_block (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_3d_pick_matrix (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_2d_pick_vector (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_1d_pick_variable (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- void starpu_ndim_filter_pick_variable (void ∗father_interface, void ∗child_interface, struct starpu_data_filter ∗f, unsigned id, unsigned nparts)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_pick_tensor_child_ops (struct starpu_data_filter ∗f, unsigned child)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_pick_block_child_ops (struct starpu_data_filter ∗f, unsigned child)

- struct starpu_data_interface_ops ∗ starpu_ndim_filter_pick_matrix_child_ops (struct starpu_data_filter ∗f, unsigned child)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_pick_vector_child_ops (struct starpu_data_filter ∗f, unsigned child)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_pick_variable_child_ops (struct starpu_data_filter ∗f, unsigned child)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_to_tensor_child_ops (struct starpu_data_filter ∗f, unsigned child)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_to_block_child_ops (struct starpu_data_filter ∗f, unsigned child)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_to_matrix_child_ops (struct starpu_data_filter ∗f, unsigned child)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_to_vector_child_ops (struct starpu_data_filter ∗f, unsigned child)
- struct starpu_data_interface_ops ∗ starpu_ndim_filter_to_variable_child_ops (struct starpu_data_filter ∗f, unsigned child)
- void starpu_filter_nparts_compute_chunk_size_and_offset (unsigned n, unsigned nparts, size_t elemsize, unsigned id, unsigned blocksize, unsigned ∗chunk_size, size_t ∗offset)

59.11 starpu_data_interfaces.h File Reference

```
#include <starpu.h>
#include <cuda_runtime.h>
#include <hip/hip_runtime.h>
```

Data Structures

- struct starpu_data_copy_methods
- struct starpu_data_interface_ops
- struct starpu_matrix_interface
- struct starpu_coo_interface
- struct starpu_block_interface
- struct starpu_tensor_interface
- struct starpu_ndim_interface
- struct starpu_vector_interface
- struct starpu_variable_interface
- struct starpu_csr_interface
- struct starpu_bcsr_interface
- struct starpu_multiformat_data_interface_ops
- struct starpu_multiformat_interface

Typedefs

- typedef cudaStream_t **starpu_cudaStream_t**
- typedef hipStream_t **starpu_hipStream_t**

Enumerations

- enum starpu_data_interface_id {
STARPU_UNKNOWN_INTERFACE_ID , STARPU_MATRIX_INTERFACE_ID , STARPU_BLOCK_INTERFACE_ID , STARPU_VECTOR_INTERFACE_ID ,
STARPU_CSR_INTERFACE_ID , STARPU_BCSR_INTERFACE_ID , STARPU_VARIABLE_INTERFACE_ID , STARPU_VOID_INTERFACE_ID ,
STARPU_MULTIFORMAT_INTERFACE_ID , STARPU_COO_INTERFACE_ID , STARPU_TENSOR_INTERFACE_ID , STARPU_NDIM_INTERFACE_ID ,
STARPU_MAX_INTERFACE_ID }

Functions

Basic API

- void starpu_data_register (starpu_data_handle_t *handleptr, int home_node, void *data_interface, struct starpu_data_interface_ops *ops)
- void starpu_data_register_ops (struct starpu_data_interface_ops *ops)
- void starpu_data_ptr_register (starpu_data_handle_t handle, unsigned node)
- void starpu_data_register_same (starpu_data_handle_t *handledst, starpu_data_handle_t handlesrc)
- void * starpu_data_handle_to_pointer (starpu_data_handle_t handle, unsigned node)
- void * starpu_data_get_local_ptr (starpu_data_handle_t handle)
- void * starpu_data_get_interface_on_node (starpu_data_handle_t handle, unsigned memory_node)
- enum starpu_data_interface_id starpu_data_get_interface_id (starpu_data_handle_t handle)
- int starpu_data_pack_node (starpu_data_handle_t handle, unsigned node, void **ptr, starpu_ssize_t *count)
- int starpu_data_pack (starpu_data_handle_t handle, void **ptr, starpu_ssize_t *count)
- int starpu_data_peek_node (starpu_data_handle_t handle, unsigned node, void *ptr, size_t count)
- int starpu_data_peek (starpu_data_handle_t handle, void *ptr, size_t count)
- int starpu_data_unpack_node (starpu_data_handle_t handle, unsigned node, void *ptr, size_t count)
- int starpu_data_unpack (starpu_data_handle_t handle, void *ptr, size_t count)
- size_t starpu_data_get_size (starpu_data_handle_t handle)
- size_t starpu_data_get_alloc_size (starpu_data_handle_t handle)
- starpu_ssize_t starpu_data_get_max_size (starpu_data_handle_t handle)
- int starpu_data_get_home_node (starpu_data_handle_t handle)
- void starpu_data_print (starpu_data_handle_t handle, unsigned node, FILE *stream)
- int starpu_data_interface_get_next_id (void)
- int starpu_interface_copy (uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, size_t dst_offset, unsigned dst_node, size_t size, void *async_data)
- int starpu_interface_copy2d (uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, size_t dst_offset, unsigned dst_node, size_t blocksize, size_t numblocks, size_t ld_src, size_t ld_dst, void *async_data)
- int starpu_interface_copy3d (uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, size_t dst_offset, unsigned dst_node, size_t blocksize, size_t numblocks1, size_t ld1_src, size_t ld1_dst, size_t numblocks2, size_t ld2_src, size_t ld2_dst, void *async_data)
- int starpu_interface_copy4d (uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, size_t dst_offset, unsigned dst_node, size_t blocksize, size_t numblocks1, size_t ld1_src, size_t ld1_dst, size_t numblocks2, size_t ld2_src, size_t ld2_dst, size_t numblocks3, size_t ld3_src, size_t ld3_dst, void *async_data)
- int starpu_interface_copynd (uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, size_t dst_offset, unsigned dst_node, size_t elemsize, size_t ndim, uint32_t *nn, uint32_t *ldn_src, uint32_t *ldn_dst, void *async_data)
- void starpu_interface_start_driver_copy_async (unsigned src_node, unsigned dst_node, double *start)
- void starpu_interface_end_driver_copy_async (unsigned src_node, unsigned dst_node, double start)
- void starpu_interface_data_copy (unsigned src_node, unsigned dst_node, size_t size)
- uintptr_t starpu_malloc_on_node_flags (unsigned dst_node, size_t size, int flags)
- uintptr_t starpu_malloc_on_node (unsigned dst_node, size_t size)
- void starpu_free_on_node_flags (unsigned dst_node, uintptr_t addr, size_t size, int flags)
- void starpu_free_on_node (unsigned dst_node, uintptr_t addr, size_t size)
- void starpu_malloc_on_node_set_default_flags (unsigned node, int flags)

MAP API

- uintptr_t starpu_interface_map (uintptr_t src, size_t src_offset, unsigned src_node, unsigned dst_node, size_t size, int *ret)
- int starpu_interface_unmap (uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, unsigned dst_node, size_t size)
- int starpu_interface_update_map (uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, size_t dst_offset, unsigned dst_node, size_t size)

Accessing Matrix Data Interfaces

- #define STARPU_MATRIX_GET_PTR(interface)
- #define STARPU_MATRIX_GET_DEV_HANDLE(interface)
- #define STARPU_MATRIX_GET_OFFSET(interface)
- #define STARPU_MATRIX_GET_NX(interface)
- #define STARPU_MATRIX_GET_NY(interface)
- #define STARPU_MATRIX_GET_LD(interface)
- #define STARPU_MATRIX_GET_ELEMSIZE(interface)
- #define STARPU_MATRIX_GET_ALLOCSIZE(interface)
- #define STARPU_MATRIX_SET_NX(interface, newnx)
- #define STARPU_MATRIX_SET_NY(interface, newny)
- #define STARPU_MATRIX_SET_LD(interface, newld)
- struct starpu_data_interface_ops **starpu_interface_matrix_ops**
- void starpu_matrix_data_register (starpu_data_handle_t ∗handle, int home_node, uintptr_t ptr, uint32_t ld, uint32_t nx, uint32_t ny, size_t elemsize)
- void starpu_matrix_data_register_allocsize (starpu_data_handle_t ∗handle, int home_node, uintptr_t ptr, uint32_t ld, uint32_t nx, uint32_t ny, size_t elemsize, size_t allocsize)
- void starpu_matrix_ptr_register (starpu_data_handle_t handle, unsigned node, uintptr_t ptr, uintptr_t dev_handle, size_t offset, uint32_t ld)
- uint32_t starpu_matrix_get_nx (starpu_data_handle_t handle)
- uint32_t starpu_matrix_get_ny (starpu_data_handle_t handle)
- uint32_t starpu_matrix_get_local_ld (starpu_data_handle_t handle)
- uintptr_t starpu_matrix_get_local_ptr (starpu_data_handle_t handle)
- size_t starpu_matrix_get_elemsize (starpu_data_handle_t handle)
- size_t starpu_matrix_get_allocsize (starpu_data_handle_t handle)

Accessing COO Data Interfaces

- #define STARPU_COO_GET_COLUMNS(interface)
- #define STARPU_COO_GET_COLUMNS_DEV_HANDLE(interface)
- #define STARPU_COO_GET_ROWS(interface)
- #define STARPU_COO_GET_ROWS_DEV_HANDLE(interface)
- #define STARPU_COO_GET_VALUES(interface)
- #define STARPU_COO_GET_VALUES_DEV_HANDLE(interface)
- #define STARPU_COO_GET_OFFSET
- #define STARPU_COO_GET_NX(interface)
- #define STARPU_COO_GET_NY(interface)
- #define STARPU_COO_GET_NVALUES(interface)
- #define STARPU_COO_GET_ELEMSIZE(interface)
- struct starpu_data_interface_ops **starpu_interface_coo_ops**
- void starpu_coo_data_register (starpu_data_handle_t ∗handleptr, int home_node, uint32_t nx, uint32_t ny, uint32_t n_values, uint32_t ∗columns, uint32_t ∗rows, uintptr_t values, size_t elemsize)

Block Data Interface

- #define STARPU_BLOCK_GET_PTR(interface)
- #define STARPU_BLOCK_GET_DEV_HANDLE(interface)
- #define STARPU_BLOCK_GET_OFFSET(interface)
- #define STARPU_BLOCK_GET_NX(interface)
- #define STARPU_BLOCK_GET_NY(interface)
- #define STARPU_BLOCK_GET_NZ(interface)
- #define STARPU_BLOCK_GET_LDY(interface)
- #define STARPU_BLOCK_GET_LDZ(interface)
- #define STARPU_BLOCK_GET_ELEMSIZE(interface)
- struct starpu_data_interface_ops **starpu_interface_block_ops**

- void starpu_block_data_register (starpu_data_handle_t ∗handle, int home_node, uintptr_t ptr, uint32_t ldy, uint32_t ldz, uint32_t nx, uint32_t ny, uint32_t nz, size_t elemsize)
- void starpu_block_ptr_register (starpu_data_handle_t handle, unsigned node, uintptr_t ptr, uintptr_t dev_↩ handle, size_t offset, uint32_t ldy, uint32_t ldz)
- uint32_t starpu_block_get_nx (starpu_data_handle_t handle)
- uint32_t starpu_block_get_ny (starpu_data_handle_t handle)
- uint32_t starpu_block_get_nz (starpu_data_handle_t handle)
- uint32_t starpu_block_get_local_ldy (starpu_data_handle_t handle)
- uint32_t starpu_block_get_local_ldz (starpu_data_handle_t handle)
- uintptr_t starpu_block_get_local_ptr (starpu_data_handle_t handle)
- size_t starpu_block_get_elemsize (starpu_data_handle_t handle)

Tensor Data Interface

- #define STARPU_TENSOR_GET_PTR(interface)
- #define STARPU_TENSOR_GET_DEV_HANDLE(interface)
- #define STARPU_TENSOR_GET_OFFSET(interface)
- #define STARPU_TENSOR_GET_NX(interface)
- #define STARPU_TENSOR_GET_NY(interface)
- #define STARPU_TENSOR_GET_NZ(interface)
- #define STARPU_TENSOR_GET_NT(interface)
- #define STARPU_TENSOR_GET_LDY(interface)
- #define STARPU_TENSOR_GET_LDZ(interface)
- #define STARPU_TENSOR_GET_LDT(interface)
- #define STARPU_TENSOR_GET_ELEMSIZE(interface)
- struct starpu_data_interface_ops **starpu_interface_tensor_ops**
- void starpu_tensor_data_register (starpu_data_handle_t ∗handle, int home_node, uintptr_t ptr, uint32_t ldy, uint32_t ldz, uint32_t ldt, uint32_t nx, uint32_t ny, uint32_t nz, uint32_t nt, size_t elemsize)
- void starpu_tensor_ptr_register (starpu_data_handle_t handle, unsigned node, uintptr_t ptr, uintptr_t dev_↩ handle, size_t offset, uint32_t ldy, uint32_t ldz, uint32_t ldt)
- uint32_t starpu_tensor_get_nx (starpu_data_handle_t handle)
- uint32_t starpu_tensor_get_ny (starpu_data_handle_t handle)
- uint32_t starpu_tensor_get_nz (starpu_data_handle_t handle)
- uint32_t starpu_tensor_get_nt (starpu_data_handle_t handle)
- uint32_t starpu_tensor_get_local_ldy (starpu_data_handle_t handle)
- uint32_t starpu_tensor_get_local_ldz (starpu_data_handle_t handle)
- uint32_t starpu_tensor_get_local_ldt (starpu_data_handle_t handle)
- uintptr_t starpu_tensor_get_local_ptr (starpu_data_handle_t handle)
- size_t starpu_tensor_get_elemsize (starpu_data_handle_t handle)

Ndim Array Data Interface

- #define STARPU_NDIM_GET_PTR(interface)
- #define STARPU_NDIM_GET_DEV_HANDLE(interface)
- #define STARPU_NDIM_GET_OFFSET(interface)
- #define STARPU_NDIM_GET_NN(interface)
- #define STARPU_NDIM_GET_LDN(interface)
- #define STARPU_NDIM_GET_NDIM(interface)
- #define STARPU_NDIM_GET_ELEMSIZE(interface)
- struct starpu_data_interface_ops **starpu_interface_ndim_ops**
- void starpu_ndim_data_register (starpu_data_handle_t ∗handleptr, int home_node, uintptr_t ptr, uint32_↩ t ∗ldn, uint32_t ∗nn, size_t ndim, size_t elemsize)
- void starpu_ndim_ptr_register (starpu_data_handle_t handle, unsigned node, uintptr_t ptr, uintptr_t dev_↩ handle, size_t offset, uint32_t ∗ldn)
- uint32_t ∗ starpu_ndim_get_nn (starpu_data_handle_t handle)

- uint32_t starpu_ndim_get_ni (starpu_data_handle_t handle, size_t i)
- uint32_t ∗ starpu_ndim_get_local_ldn (starpu_data_handle_t handle)
- uint32_t starpu_ndim_get_local_ldi (starpu_data_handle_t handle, size_t i)
- uintptr_t starpu_ndim_get_local_ptr (starpu_data_handle_t handle)
- size_t starpu_ndim_get_ndim (starpu_data_handle_t handle)
- size_t starpu_ndim_get_elemsize (starpu_data_handle_t handle)

Vector Data Interface

- #define STARPU_VECTOR_GET_PTR(interface)
- #define STARPU_VECTOR_GET_DEV_HANDLE(interface)
- #define STARPU_VECTOR_GET_OFFSET(interface)
- #define STARPU_VECTOR_GET_NX(interface)
- #define STARPU_VECTOR_GET_ELEMSIZE(interface)
- #define STARPU_VECTOR_GET_ALLOCSIZE(interface)
- #define STARPU_VECTOR_GET_SLICE_BASE(interface)
- #define STARPU_VECTOR_SET_NX(interface, newnx)
- struct starpu_data_interface_ops **starpu_interface_vector_ops**
- void starpu_vector_data_register (starpu_data_handle_t ∗handle, int home_node, uintptr_t ptr, uint32_t nx, size_t elemsize)
- void starpu_vector_data_register_allocsize (starpu_data_handle_t ∗handle, int home_node, uintptr_t ptr, uint32_t nx, size_t elemsize, size_t allocsize)
- void starpu_vector_ptr_register (starpu_data_handle_t handle, unsigned node, uintptr_t ptr, uintptr_t dev_handle, size_t offset)
- uint32_t starpu_vector_get_nx (starpu_data_handle_t handle)
- size_t starpu_vector_get_elemsize (starpu_data_handle_t handle)
- size_t starpu_vector_get_allocsize (starpu_data_handle_t handle)
- uintptr_t starpu_vector_get_local_ptr (starpu_data_handle_t handle)

Variable Data Interface

- #define STARPU_VARIABLE_GET_PTR(interface)
- #define STARPU_VARIABLE_GET_OFFSET(interface)
- #define STARPU_VARIABLE_GET_ELEMSIZE(interface)
- #define STARPU_VARIABLE_GET_DEV_HANDLE(interface)
- struct starpu_data_interface_ops **starpu_interface_variable_ops**
- void starpu_variable_data_register (starpu_data_handle_t ∗handle, int home_node, uintptr_t ptr, size_t size)
- void starpu_variable_ptr_register (starpu_data_handle_t handle, unsigned node, uintptr_t ptr, uintptr_t dev_handle, size_t offset)
- size_t starpu_variable_get_elemsize (starpu_data_handle_t handle)
- uintptr_t starpu_variable_get_local_ptr (starpu_data_handle_t handle)

CSR Data Interface

- #define STARPU_CSR_GET_NNZ(interface)
- #define STARPU_CSR_GET_NROW(interface)
- #define STARPU_CSR_GET_NZVAL(interface)
- #define STARPU_CSR_GET_NZVAL_DEV_HANDLE(interface)
- #define STARPU_CSR_GET_COLIND(interface)
- #define STARPU_CSR_GET_RAM_COLIND(interface)
- #define STARPU_CSR_GET_COLIND_DEV_HANDLE(interface)
- #define STARPU_CSR_GET_ROWPTR(interface)
- #define STARPU_CSR_GET_RAM_ROWPTR(interface)
- #define STARPU_CSR_GET_ROWPTR_DEV_HANDLE(interface)
- #define STARPU_CSR_GET_OFFSET

- #define STARPU_CSR_GET_FIRSTENTRY(interface)
- #define STARPU_CSR_GET_ELEMSIZE(interface)
- struct starpu_data_interface_ops **starpu_interface_csr_ops**
- void starpu_csr_data_register (starpu_data_handle_t ∗handle, int home_node, uint32_t nnz, uint32_t nrow, uintptr_t nzval, uint32_t ∗colind, uint32_t ∗rowptr, uint32_t firstentry, size_t elemsize)
- uint32_t starpu_csr_get_nnz (starpu_data_handle_t handle)
- uint32_t starpu_csr_get_nrow (starpu_data_handle_t handle)
- uint32_t starpu_csr_get_firstentry (starpu_data_handle_t handle)
- uintptr_t starpu_csr_get_local_nzval (starpu_data_handle_t handle)
- uint32_t ∗ starpu_csr_get_local_colind (starpu_data_handle_t handle)
- uint32_t ∗ starpu_csr_get_local_rowptr (starpu_data_handle_t handle)
- size_t starpu_csr_get_elemsize (starpu_data_handle_t handle)

BCSR Data Interface

- #define STARPU_BCSR_GET_NNZ(interface)
- #define STARPU_BCSR_GET_NROW(interface)
- #define STARPU_BCSR_GET_NZVAL(interface)
- #define STARPU_BCSR_GET_NZVAL_DEV_HANDLE(interface)
- #define STARPU_BCSR_GET_COLIND(interface)
- #define STARPU_BCSR_GET_RAM_COLIND(interface)
- #define STARPU_BCSR_GET_COLIND_DEV_HANDLE(interface)
- #define STARPU_BCSR_GET_ROWPTR(interface)
- #define STARPU_BCSR_GET_RAM_ROWPTR(interface)
- #define STARPU_BCSR_GET_ROWPTR_DEV_HANDLE(interface)
- #define STARPU_BCSR_GET_FIRSTENTRY(interface)
- #define STARPU_BCSR_GET_R(interface)
- #define STARPU_BCSR_GET_C(interface)
- #define STARPU_BCSR_GET_ELEMSIZE(interface)
- #define STARPU_BCSR_GET_OFFSET
- struct starpu_data_interface_ops **starpu_interface_bcsr_ops**
- void starpu_bcsr_data_register (starpu_data_handle_t ∗handle, int home_node, uint32_t nnz, uint32_t nrow, uintptr_t nzval, uint32_t ∗colind, uint32_t ∗rowptr, uint32_t firstentry, uint32_t r, uint32_t c, size_t elemsize)
- uint32_t starpu_bcsr_get_nnz (starpu_data_handle_t handle)
- uint32_t starpu_bcsr_get_nrow (starpu_data_handle_t handle)
- uint32_t starpu_bcsr_get_firstentry (starpu_data_handle_t handle)
- uintptr_t starpu_bcsr_get_local_nzval (starpu_data_handle_t handle)
- uint32_t ∗ starpu_bcsr_get_local_colind (starpu_data_handle_t handle)
- uint32_t ∗ starpu_bcsr_get_local_rowptr (starpu_data_handle_t handle)
- uint32_t starpu_bcsr_get_r (starpu_data_handle_t handle)
- uint32_t starpu_bcsr_get_c (starpu_data_handle_t handle)
- size_t starpu_bcsr_get_elemsize (starpu_data_handle_t handle)

Multiformat Data Interface

- #define STARPU_MULTIFORMAT_GET_CPU_PTR(interface)
- #define STARPU_MULTIFORMAT_GET_CUDA_PTR(interface)
- #define STARPU_MULTIFORMAT_GET_HIP_PTR(interface)
- #define STARPU_MULTIFORMAT_GET_OPENCL_PTR(interface)
- #define STARPU_MULTIFORMAT_GET_NX(interface)
- void starpu_multiformat_data_register (starpu_data_handle_t ∗handle, int home_node, void ∗ptr, uint32_t nobjects, struct starpu_multiformat_data_interface_ops ∗format_ops)

Void Data Interface

- struct starpu_data_interface_ops **starpu_interface_void_ops**
- void starpu_void_data_register (starpu_data_handle_t ∗handle)

59.12 starpu_deprecated_api.h File Reference

Macros

- #define **starpu_permodel_history_based_expected_perf**

59.13 starpu_disk.h File Reference

```
#include <sys/types.h>
#include <starpu_config.h>
```

Data Structures

- struct starpu_disk_ops

Macros

- #define STARPU_DISK_SIZE_MIN

Functions

- void starpu_disk_close (unsigned node, void ∗obj, size_t size)
- void ∗ starpu_disk_open (unsigned node, void ∗pos, size_t size)
- int starpu_disk_register (struct starpu_disk_ops ∗func, void ∗parameter, starpu_ssize_t size)

Variables

- struct starpu_disk_ops starpu_disk_stdio_ops
- struct starpu_disk_ops starpu_disk_hdf5_ops
- struct starpu_disk_ops starpu_disk_unistd_ops
- struct starpu_disk_ops starpu_disk_unistd_o_direct_ops
- struct starpu_disk_ops starpu_disk_leveldb_ops
- int starpu_disk_swap_node

59.14 starpu_driver.h File Reference

```
#include <starpu_config.h>
#include <starpu_opencl.h>
#include <starpu_max_fpga.h>
```

Data Structures

- struct starpu_driver
- union starpu_driver.id

Functions

- void starpu_drivers_preinit (void)
- int starpu_driver_run (struct starpu_driver ∗d)
- void starpu_drivers_request_termination (void)
- int starpu_driver_init (struct starpu_driver ∗d)
- int starpu_driver_run_once (struct starpu_driver ∗d)
- int starpu_driver_deinit (struct starpu_driver ∗d)

59.15 starpu_expert.h File Reference

Functions

- void starpu_wake_all_blocked_workers (void)
- int starpu_progression_hook_register (unsigned(∗func)(void ∗arg), void ∗arg)
- void starpu_progression_hook_deregister (int hook_id)
- int **starpu_idle_hook_register** (unsigned(∗func)(void ∗arg), void ∗arg)
- void **starpu_idle_hook_deregister** (int hook_id)

59.16 starpu_fxt.h File Reference

```
#include <starpu_config.h>
#include <starpu_perfmodel.h>
```

Data Structures

- struct starpu_fxt_codelet_event
- struct starpu_fxt_mpi_offset
- struct starpu_fxt_options

Functions

- void **starpu_fxt_options_init** (struct starpu_fxt_options ∗options)
- void **starpu_fxt_options_shutdown** (struct starpu_fxt_options ∗options)
- void **starpu_fxt_generate_trace** (struct starpu_fxt_options ∗options)
- void starpu_fxt_autostart_profiling (int autostart)
- void starpu_fxt_start_profiling (void)
- void starpu_fxt_stop_profiling (void)
- void **starpu_fxt_write_data_trace** (char ∗filename_in)
- void **starpu_fxt_write_data_trace_in_dir** (char ∗filename_in, char ∗dir)
- int starpu_fxt_is_enabled (void)
- void starpu_fxt_trace_user_event (unsigned long code)
- void starpu_fxt_trace_user_event_string (const char ∗s)

59.17 starpu_hash.h File Reference

```
#include <stdint.h>
#include <stddef.h>
```

Functions

- uint32_t starpu_hash_crc32c_be_n (const void ∗input, size_t n, uint32_t inputcrc)
- uint32_t starpu_hash_crc32c_be_ptr (void ∗input, uint32_t inputcrc)
- uint32_t starpu_hash_crc32c_be (uint32_t input, uint32_t inputcrc)
- uint32_t starpu_hash_crc32c_string (const char ∗str, uint32_t inputcrc)

59.18 starpu_helper.h File Reference

```
#include <stdio.h>
#include <starpu.h>
#include <hwloc.h>
```

Macros

- #define STARPU_MIN(a, b)
- #define STARPU_MAX(a, b)
- #define STARPU_POISON_PTR
- #define starpu_getenv_string_var_default(s, ss, d)
- #define starpu_getenv_size_default(s, d)
- #define starpu_getenv_number(s)
- #define starpu_getenv_number_default(s, d)
- #define starpu_getenv_float_default(s, d)

Functions

- char ∗ starpu_getenv (const char ∗str)
- int starpu_get_env_string_var_default (const char ∗str, const char ∗strings[ ], int defvalue)
- int starpu_get_env_size_default (const char ∗str, int defval)
- static __starpu_inline int starpu_get_env_number (const char ∗str)
- static __starpu_inline int **starpu_get_env_number_default** (const char ∗str, int defval)
- static __starpu_inline float **starpu_get_env_float_default** (const char ∗str, float defval)
- void starpu_execute_on_each_worker (void(∗func)(void ∗), void ∗arg, uint32_t where)
- void starpu_execute_on_each_worker_ex (void(∗func)(void ∗), void ∗arg, uint32_t where, const char ∗name)
- void starpu_execute_on_specific_workers (void(∗func)(void ∗), void ∗arg, unsigned num_workers, unsigned ∗workers, const char ∗name)
- double starpu_timing_now (void)
- int starpu_data_cpy (starpu_data_handle_t dst_handle, starpu_data_handle_t src_handle, int asynchronous, void(∗callback_func)(void ∗), void ∗callback_arg)
- int starpu_data_cpy_priority (starpu_data_handle_t dst_handle, starpu_data_handle_t src_handle, int asynchronous, void(∗callback_func)(void ∗), void ∗callback_arg, int priority)
- int starpu_data_dup_ro (starpu_data_handle_t ∗dst_handle, starpu_data_handle_t src_handle, int asynchronous)
- void starpu_display_bindings (void)
- int starpu_get_pu_os_index (unsigned logical_index)
- long starpu_get_memory_location_bitmap (void ∗ptr, size_t size)
- hwloc_topology_t starpu_get_hwloc_topology (void)

Variables

- int **_starpu_silent**

59.19 starpu_heteroprio.h File Reference

`#include <starpu.h>`

Macros

- #define **STARPU_HETEROPRIO_MAX_PREFETCH**
- #define **STARPU_AUTOHETEROPRIO_PRIORITY_ORDERING_POLICY_COUNT**

Enumerations

- enum starpu_autoheteroprio_priority_ordering_policy {
STARPU_HETEROPRIO_NOD_TIME_COMBINATION , **STARPU_HETEROPRIO_BEST_NODS_SCORE** , **STARPU_HETEROPRIO_BEST_NODS** , **STARPU_HETEROPRIO_URT_PURE** ,
STARPU_HETEROPRIO_URT , **STARPU_HETEROPRIO_URT_2** , **STARPU_HETEROPRIO_URT_DOT_DIFF_PURE** , **STARPU_HETEROPRIO_URT_DOT_DIFF_PURE_2** ,
STARPU_HETEROPRIO_URT_DOT_REL_DIFF_PURE , **STARPU_HETEROPRIO_URT_DOT_REL_DIFF_PURE_2** , **STARPU_HETEROPRIO_URT_DOT_DIFF_2** , **STARPU_HETEROPRIO_URT_DOT_DIFF_3** ,
STARPU_HETEROPRIO_URT_DOT_DIFF_4 , **STARPU_HETEROPRIO_URT_DOT_DIFF_5** , **STARPU_HETEROPRIO_URT_DOT_DIFF_6** , **STARPU_HETEROPRIO_URT_DOT_DIFF_7** ,
STARPU_HETEROPRIO_URT_DOT_DIFF_8 , **STARPU_HETEROPRIO_URT_DOT_DIFF_9** , **STARPU_HETEROPRIO_URT_DOT_DIFF_10** , **STARPU_HETEROPRIO_URT_DOT_DIFF_11** ,
STARPU_HETEROPRIO_URTS_PER_SECONDS , **STARPU_HETEROPRIO_URTS_PER_SECONDS_2** , **STARPU_HETEROPRIO_URTS_PER_SECONDS_DIFF** , **STARPU_HETEROPRIO_URTS_TIME_RELEASED_DIFF** ,
STARPU_HETEROPRIO_URTS_TIME_COMBINATION , **STARPU_HETEROPRIO_NODS_PER_SECOND** , **STARPU_HETEROPRIO_NODS_TIME_RELEASED** , **STARPU_HETEROPRIO_NODS_TIME_RELEASED_DIFF** }

Functions

- void starpu_heteroprio_set_use_locality (unsigned sched_ctx_id, unsigned use_locality)
- void starpu_heteroprio_set_nb_prios (unsigned sched_ctx_id, enum starpu_worker_archtype arch, unsigned max_prio)
- void starpu_heteroprio_set_mapping (unsigned sched_ctx_id, enum starpu_worker_archtype arch, unsigned source_prio, unsigned dest_bucket_id)
- void starpu_heteroprio_set_faster_arch (unsigned sched_ctx_id, enum starpu_worker_archtype arch, unsigned bucket_id)
- void starpu_heteroprio_set_arch_slow_factor (unsigned sched_ctx_id, enum starpu_worker_archtype arch, unsigned bucket_id, float slow_factor)
- void starpu_heteroprio_map_wgroup_memory_nodes (unsigned sched_ctx_id)
- void starpu_heteroprio_print_wgroups (FILE ∗stream, unsigned sched_ctx_id)

Variables

- static const char **starpu_autoheteroprio_priority_ordering_policy_names** [STARPU_AUTOHETEROPRIO_PRIORITY_ORDERING_POLICY_COUNT][64]

59.20 starpu_hip.h File Reference

```
#include <starpu_config.h>
#include <hip/hip_runtime.h>
#include <hip/hip_runtime_api.h>
```

Macros

- #define STARPU_HIPBLAS_REPORT_ERROR(status)
- #define STARPU_HIP_REPORT_ERROR(status)

Functions

- void starpu_hipblas_report_error (const char ∗func, const char ∗file, int line, int status)
- void starpu_hip_report_error (const char ∗func, const char ∗file, int line, hipError_t status)
- hipStream_t starpu_hip_get_local_stream (void)
- const struct hipDeviceProp_t ∗ starpu_hip_get_device_properties (unsigned workerid)
- int starpu_hip_copy_async_sync (void ∗src_ptr, unsigned src_node, void ∗dst_ptr, unsigned dst_node, size_t ssize, hipStream_t stream, hipMemcpyKind kind)
- int starpu_hip_copy2d_async_sync (void ∗src_ptr, unsigned src_node, void ∗dst_ptr, unsigned dst_node, size_t blocksize, size_t numblocks, size_t ld_src, size_t ld_dst, hipStream_t stream, hipMemcpyKind kind)
- int starpu_hip_copy3d_async_sync (void ∗src_ptr, unsigned src_node, void ∗dst_ptr, unsigned dst_node, size_t blocksize, size_t numblocks_1, size_t ld1_src, size_t ld1_dst, size_t numblocks_2, size_t ld2_src, size_t ld2_dst, hipStream_t stream, hipMemcpyKind kind)
- void starpu_hip_set_device (int devid)

59.21 starpu_scheduler_toolbox.h File Reference

```
#include <starpu.h>
#include <starpu_scheduler.h>
```

Typedefs

- typedef struct starpu_st_fifo_taskq ∗ starpu_st_fifo_taskq_t
- typedef struct starpu_st_prio_deque ∗ starpu_st_prio_deque_t

Functions

- starpu_st_fifo_taskq_t starpu_st_fifo_taskq_create (void) STARPU_ATTRIBUTE_MALLOC
- void **starpu_st_fifo_taskq_init** (starpu_st_fifo_taskq_t fifo)
- void **starpu_st_fifo_taskq_destroy** (starpu_st_fifo_taskq_t fifo)
- int **starpu_st_fifo_taskq_empty** (starpu_st_fifo_taskq_t fifo)
- double **starpu_st_fifo_taskq_get_exp_len_prev_task_list** (starpu_st_fifo_taskq_t fifo_queue, struct starpu_task ∗task, int workerid, int nimpl, int ∗fifo_ntasks)
- unsigned starpu_st_fifo_ntasks_get (starpu_st_fifo_taskq_t fifo)
- void starpu_st_fifo_ntasks_inc (starpu_st_fifo_taskq_t fifo, int n)
- unsigned ∗ starpu_st_fifo_ntasks_per_priority_get (starpu_st_fifo_taskq_t fifo)
- unsigned starpu_st_fifo_nprocessed_get (starpu_st_fifo_taskq_t fifo)
- void starpu_st_fifo_nprocessed_inc (starpu_st_fifo_taskq_t fifo, int n)
- double starpu_st_fifo_exp_start_get (starpu_st_fifo_taskq_t fifo)
- void starpu_st_fifo_exp_start_set (starpu_st_fifo_taskq_t fifo, double exp_start)
- double starpu_st_fifo_exp_end_get (starpu_st_fifo_taskq_t fifo)
- void starpu_st_fifo_exp_end_set (starpu_st_fifo_taskq_t fifo, double exp_end)
- double starpu_st_fifo_exp_len_get (starpu_st_fifo_taskq_t fifo)
- void starpu_st_fifo_exp_len_set (starpu_st_fifo_taskq_t fifo, double exp_len)
- void starpu_st_fifo_exp_len_inc (starpu_st_fifo_taskq_t fifo, double exp_len)
- double ∗ starpu_st_fifo_exp_len_per_priority_get (starpu_st_fifo_taskq_t fifo)
- double starpu_st_fifo_pipeline_len_get (starpu_st_fifo_taskq_t fifo)
- void starpu_st_fifo_pipeline_len_set (starpu_st_fifo_taskq_t fifo, double pipeline_len)
- void starpu_st_fifo_pipeline_len_inc (starpu_st_fifo_taskq_t fifo, double pipeline_len)
- int **starpu_st_fifo_taskq_push_sorted_task** (starpu_st_fifo_taskq_t fifo_queue, struct starpu_task ∗task)

- int **starpu_st_fifo_taskq_push_task** (starpu_st_fifo_taskq_t fifo, struct starpu_task ∗task)
- int **starpu_st_fifo_taskq_push_back_task** (starpu_st_fifo_taskq_t fifo_queue, struct starpu_task ∗task)
- int **starpu_st_fifo_taskq_pop_this_task** (starpu_st_fifo_taskq_t fifo_queue, int workerid, struct starpu_task ∗task)
- struct starpu_task ∗ **starpu_st_fifo_taskq_pop_task** (starpu_st_fifo_taskq_t fifo, int workerid)
- struct starpu_task ∗ starpu_st_fifo_taskq_pop_local_task (starpu_st_fifo_taskq_t fifo)
- struct starpu_task ∗ starpu_st_fifo_taskq_pop_first_ready_task (starpu_st_fifo_taskq_t fifo_queue, unsigned workerid, int num_priorities)
- void starpu_st_prio_deque_init (starpu_st_prio_deque_t pdeque)
- void **starpu_st_prio_deque_destroy** (starpu_st_prio_deque_t pdeque)
- int starpu_st_prio_deque_is_empty (starpu_st_prio_deque_t pdeque)
- int **starpu_st_prio_deque_push_back_task** (starpu_st_prio_deque_t pdeque, struct starpu_task ∗task)
- int starpu_st_prio_deque_push_front_task (starpu_st_prio_deque_t pdeque, struct starpu_task ∗task)
- struct starpu_task ∗ starpu_st_prio_deque_pop_task_for_worker (starpu_st_prio_deque_t pdeque, int workerid, struct starpu_task ∗∗skipped)
- struct starpu_task ∗ starpu_st_prio_deque_deque_task_for_worker (starpu_st_prio_deque_t pdeque, int workerid, struct starpu_task ∗∗skipped)
- struct starpu_task ∗ **starpu_st_prio_deque_deque_first_ready_task** (starpu_st_prio_deque_t pdeque, unsigned workerid)
- struct starpu_task ∗ **starpu_st_prio_deque_pop_task** (starpu_st_prio_deque_t pdeque)
- struct starpu_task ∗ **starpu_st_prio_deque_highest_task** (starpu_st_prio_deque_t pdeque)
- struct starpu_task ∗ **starpu_st_prio_deque_pop_back_task** (starpu_st_prio_deque_t pdeque)
- int **starpu_st_prio_deque_pop_this_task** (starpu_st_prio_deque_t pdeque, int workerid, struct starpu_task ∗task)
- void **starpu_st_prio_deque_erase** (starpu_st_prio_deque_t pdeque, struct starpu_task ∗task)
- int **starpu_st_normalize_prio** (int priority, int num_priorities, unsigned sched_ctx_id)
- int **starpu_st_non_ready_buffers_count** (struct starpu_task ∗task, unsigned worker)
- void **starpu_st_non_ready_buffers_size** (struct starpu_task ∗task, unsigned worker, size_t ∗non_readyp, size_t ∗non_loadingp, size_t ∗non_allocatedp)

59.22 starpu_max_fpga.h File Reference

```
#include <starpu_config.h>
#include <MaxSLiCInterface.h>
```

Data Structures

- struct starpu_max_load

Functions

- max_engine_t ∗ starpu_max_fpga_get_local_engine (void)

59.23 starpu_mod.f90 File Reference

Data Types

- interface starpu_mod::starpu_conf_init
- interface starpu_mod::starpu_init
- interface starpu_mod::starpu_pause
- interface starpu_mod::starpu_resume
- interface starpu_mod::starpu_shutdown
- interface starpu_mod::starpu_asynchronous_copy_disabled
- interface starpu_mod::starpu_asynchronous_cuda_copy_disabled

- interface starpu_mod::starpu_asynchronous_opencl_copy_disabled
- interface starpu_mod::starpu_display_stats
- interface starpu_mod::starpu_get_version
- interface starpu_mod::starpu_cpu_worker_get_count
- interface starpu_mod::starpu_task_wait_for_all

59.24 starpu_mpi.h File Reference

```
#include <starpu.h>
#include <mpi.h>
#include <starpu_mpi_ft.h>
#include <stdint.h>
```

Data Structures

- struct starpu_mpi_task_exchange_params

Functions

Communication Cache

- int starpu_mpi_cache_is_enabled (void)
- int starpu_mpi_cache_set (int enabled)
- void starpu_mpi_cache_flush (MPI_Comm comm, starpu_data_handle_t data_handle)
- void starpu_mpi_cache_flush_all_data (MPI_Comm comm)
- int starpu_mpi_cached_receive (starpu_data_handle_t data_handle)
- int starpu_mpi_cached_receive_set (starpu_data_handle_t data)
- int **starpu_mpi_cached_cp_receive_set** (starpu_data_handle_t data_handle)
- void starpu_mpi_cached_receive_clear (starpu_data_handle_t data)
- int starpu_mpi_cached_send (starpu_data_handle_t data_handle, int dest)
- int starpu_mpi_cached_send_set (starpu_data_handle_t data, int dest)
- void starpu_mpi_cached_send_clear (starpu_data_handle_t data)

Collective Operations

- int starpu_mpi_redux_data (MPI_Comm comm, starpu_data_handle_t data_handle)
- int starpu_mpi_redux_data_prio (MPI_Comm comm, starpu_data_handle_t data_handle, int prio)
- int starpu_mpi_redux_data_tree (MPI_Comm comm, starpu_data_handle_t data_handle, int arity)
- int starpu_mpi_redux_data_prio_tree (MPI_Comm comm, starpu_data_handle_t data_handle, int prio, int arity)
- int starpu_mpi_scatter_detached (starpu_data_handle_t ∗data_handles, int count, int root, MPI_Comm comm, void(∗scallback)(void ∗), void ∗sarg, void(∗rcallback)(void ∗), void ∗rarg)
- int starpu_mpi_gather_detached (starpu_data_handle_t ∗data_handles, int count, int root, MPI_Comm comm, void(∗scallback)(void ∗), void ∗sarg, void(∗rcallback)(void ∗), void ∗rarg)

Dynamic Broadcasts

- void starpu_mpi_coop_sends_set_use (int use_coop_sends)
- int starpu_mpi_coop_sends_get_use (void)
- void starpu_mpi_coop_sends_data_handle_nb_sends (starpu_data_handle_t data_handle, int nb_sends)

Statistics

- void starpu_mpi_comm_stats_disable (void)
- void starpu_mpi_comm_stats_enable (void)
- void starpu_mpi_comm_stats_retrieve (size_t ∗comm_stats)

Miscellaneous

- int **starpu_mpi_pre_submit_hook_register** (void(∗f)(struct starpu_task ∗))

- int **starpu_mpi_pre_submit_hook_unregister** (void)
- int starpu_mpi_data_cpy (starpu_data_handle_t dst_handle, starpu_data_handle_t src_handle, MPI_↩ Comm comm, int asynchronous, void(∗callback_func)(void ∗), void ∗callback_arg)
- int starpu_mpi_data_cpy_priority (starpu_data_handle_t dst_handle, starpu_data_handle_t src_handle, MPI_Comm comm, int asynchronous, void(∗callback_func)(void ∗), void ∗callback_arg, int priority)

Data Tags Management

- int64_t starpu_mpi_tags_allocate (int64_t nbtags)
- void starpu_mpi_tags_free (int64_t mintag)

Initialisation

- #define STARPU_MPI_TAG_UB
- int starpu_mpi_init_conf (int ∗argc, char ∗∗∗argv, int initialize_mpi, MPI_Comm comm, struct starpu_conf ∗conf)
- int starpu_mpi_init_comm (int ∗argc, char ∗∗∗argv, int initialize_mpi, MPI_Comm comm)
- int starpu_mpi_init (int ∗argc, char ∗∗∗argv, int initialize_mpi)
- int starpu_mpi_initialize (void)
- int starpu_mpi_initialize_extended (int ∗rank, int ∗world_size)
- int starpu_mpi_shutdown (void)
- int starpu_mpi_shutdown_comm (MPI_Comm comm)
- int starpu_mpi_comm_register (MPI_Comm comm)
- int starpu_mpi_comm_size (MPI_Comm comm, int ∗size)
- int starpu_mpi_comm_rank (MPI_Comm comm, int ∗rank)
- int starpu_mpi_world_rank (void)
- int starpu_mpi_world_size (void)
- int starpu_mpi_comm_get_attr (MPI_Comm comm, int keyval, void ∗attribute_val, int ∗flag)
- int starpu_mpi_get_thread_cpuid (void)
- int starpu_mpi_get_communication_tag (void)
- void starpu_mpi_set_communication_tag (int tag)

MPI Insert Task

- #define STARPU_MPI_PER_NODE
- #define starpu_mpi_data_register(data_handle, data_tag, rank)
- #define starpu_data_set_tag
- #define starpu_mpi_data_set_rank(handle, rank)
- #define starpu_data_set_rank
- #define starpu_data_get_rank
- #define starpu_data_get_tag
- void starpu_mpi_data_register_comm (starpu_data_handle_t data_handle, starpu_mpi_tag_t data_tag, int rank, MPI_Comm comm)
- void starpu_mpi_data_set_tag (starpu_data_handle_t handle, starpu_mpi_tag_t data_tag)
- void starpu_mpi_data_set_rank_comm (starpu_data_handle_t handle, int rank, MPI_Comm comm)
- int starpu_mpi_data_get_rank (starpu_data_handle_t handle)
- starpu_mpi_tag_t starpu_mpi_data_get_tag (starpu_data_handle_t handle)
- char ∗ starpu_mpi_data_get_redux_map (starpu_data_handle_t handle)
- int starpu_mpi_task_insert (MPI_Comm comm, struct starpu_codelet ∗codelet,...)
- int starpu_mpi_insert_task (MPI_Comm comm, struct starpu_codelet ∗codelet,...)
- struct starpu_task ∗ starpu_mpi_task_build (MPI_Comm comm, struct starpu_codelet ∗codelet,...)
- struct starpu_task ∗ starpu_mpi_task_build_v (MPI_Comm comm, struct starpu_codelet ∗codelet, va_list varg_list)
- int starpu_mpi_task_post_build (MPI_Comm comm, struct starpu_codelet ∗codelet,...)
- int starpu_mpi_task_post_build_v (MPI_Comm comm, struct starpu_codelet ∗codelet, va_list varg_list)
- int starpu_mpi_task_exchange_data_before_execution (MPI_Comm comm, struct starpu_task ∗task, struct starpu_data_descr ∗descrs, struct starpu_mpi_task_exchange_params ∗params)

- int starpu_mpi_task_exchange_data_after_execution (MPI_Comm comm, struct starpu_data_descr ∗descrs, unsigned nb_data, struct starpu_mpi_task_exchange_params params)
- int starpu_mpi_get_data_on_node (MPI_Comm comm, starpu_data_handle_t data_handle, int node)
- int starpu_mpi_get_data_on_node_detached (MPI_Comm comm, starpu_data_handle_t data_handle, int node, void(∗callback)(void ∗), void ∗arg)
- void starpu_mpi_get_data_on_all_nodes_detached (MPI_Comm comm, starpu_data_handle_t data_handle)
- void starpu_mpi_data_migrate (MPI_Comm comm, starpu_data_handle_t handle, int new_rank)

Node Selection Policy

- #define STARPU_MPI_NODE_SELECTION_CURRENT_POLICY
- #define STARPU_MPI_NODE_SELECTION_MOST_R_DATA
- typedef int(∗ **starpu_mpi_select_node_policy_func_t**) (int me, int nb_nodes, struct starpu_data_descr ∗descr, int nb_data)
- int starpu_mpi_node_selection_register_policy (starpu_mpi_select_node_policy_func_t policy_func)
- int starpu_mpi_node_selection_unregister_policy (int policy)
- int starpu_mpi_node_selection_get_current_policy (void)
- int starpu_mpi_node_selection_set_current_policy (int policy)

Communication

- typedef void ∗ starpu_mpi_req
- typedef int64_t starpu_mpi_tag_t
- typedef int(∗ **starpu_mpi_datatype_allocate_func_t**) (starpu_data_handle_t, MPI_Datatype ∗)
- typedef int(∗ **starpu_mpi_datatype_node_allocate_func_t**) (starpu_data_handle_t, unsigned node, MPI_Datatype ∗)
- typedef void(∗ **starpu_mpi_datatype_free_func_t**) (MPI_Datatype ∗)
- int starpu_mpi_isend (starpu_data_handle_t data_handle, starpu_mpi_req ∗req, int dest, starpu_mpi_tag_t data_tag, MPI_Comm comm)
- int starpu_mpi_isend_prio (starpu_data_handle_t data_handle, starpu_mpi_req ∗req, int dest, starpu_mpi_tag_t data_tag, int prio, MPI_Comm comm)
- int starpu_mpi_irecv (starpu_data_handle_t data_handle, starpu_mpi_req ∗req, int source, starpu_mpi_tag_t data_tag, MPI_Comm comm)
- int starpu_mpi_send (starpu_data_handle_t data_handle, int dest, starpu_mpi_tag_t data_tag, MPI_Comm comm)
- int starpu_mpi_send_prio (starpu_data_handle_t data_handle, int dest, starpu_mpi_tag_t data_tag, int prio, MPI_Comm comm)
- int starpu_mpi_recv (starpu_data_handle_t data_handle, int source, starpu_mpi_tag_t data_tag, MPI_Comm comm, MPI_Status ∗status)
- int starpu_mpi_recv_prio (starpu_data_handle_t data_handle, int source, starpu_mpi_tag_t data_tag, int prio, MPI_Comm comm, MPI_Status ∗status)
- int starpu_mpi_isend_detached (starpu_data_handle_t data_handle, int dest, starpu_mpi_tag_t data_tag, MPI_Comm comm, void(∗callback)(void ∗), void ∗arg)
- int starpu_mpi_isend_detached_prio (starpu_data_handle_t data_handle, int dest, starpu_mpi_tag_t data_tag, int prio, MPI_Comm comm, void(∗callback)(void ∗), void ∗arg)
- int starpu_mpi_irecv_detached (starpu_data_handle_t data_handle, int source, starpu_mpi_tag_t data_tag, MPI_Comm comm, void(∗callback)(void ∗), void ∗arg)
- int starpu_mpi_irecv_detached_prio (starpu_data_handle_t data_handle, int source, starpu_mpi_tag_t data_tag, int prio, MPI_Comm comm, void(∗callback)(void ∗), void ∗arg)
- int starpu_mpi_irecv_detached_sequential_consistency (starpu_data_handle_t data_handle, int source, starpu_mpi_tag_t data_tag, MPI_Comm comm, void(∗callback)(void ∗), void ∗arg, int sequential_consistency)
- int starpu_mpi_issend (starpu_data_handle_t data_handle, starpu_mpi_req ∗req, int dest, starpu_mpi_tag_t data_tag, MPI_Comm comm)
- int starpu_mpi_issend_prio (starpu_data_handle_t data_handle, starpu_mpi_req ∗req, int dest, starpu_mpi_tag_t data_tag, int prio, MPI_Comm comm)

- int starpu_mpi_issend_detached (starpu_data_handle_t data_handle, int dest, starpu_mpi_tag_t data_tag, MPI_Comm comm, void(∗callback)(void ∗), void ∗arg)
- int starpu_mpi_issend_detached_prio (starpu_data_handle_t data_handle, int dest, starpu_mpi_tag_t data_tag, int prio, MPI_Comm comm, void(∗callback)(void ∗), void ∗arg)
- int starpu_mpi_wait (starpu_mpi_req ∗req, MPI_Status ∗status)
- int starpu_mpi_test (starpu_mpi_req ∗req, int ∗flag, MPI_Status ∗status)
- int starpu_mpi_barrier (MPI_Comm comm)
- int starpu_mpi_wait_for_all (MPI_Comm comm)
- int starpu_mpi_isend_detached_unlock_tag (starpu_data_handle_t data_handle, int dest, starpu_mpi_tag_t data_tag, MPI_Comm comm, starpu_tag_t tag)
- int starpu_mpi_isend_detached_unlock_tag_prio (starpu_data_handle_t data_handle, int dest, starpu_mpi_tag_t data_tag, int prio, MPI_Comm comm, starpu_tag_t tag)
- int starpu_mpi_irecv_detached_unlock_tag (starpu_data_handle_t data_handle, int source, starpu_mpi_tag_t data_tag, MPI_Comm comm, starpu_tag_t tag)
- int starpu_mpi_isend_array_detached_unlock_tag (unsigned array_size, starpu_data_handle_t ∗data_handle, int ∗dest, starpu_mpi_tag_t ∗data_tag, MPI_Comm ∗comm, starpu_tag_t tag)
- int starpu_mpi_isend_array_detached_unlock_tag_prio (unsigned array_size, starpu_data_handle_t ∗data_handle, int ∗dest, starpu_mpi_tag_t ∗data_tag, int ∗prio, MPI_Comm ∗comm, starpu_tag_t tag)
- int starpu_mpi_irecv_array_detached_unlock_tag (unsigned array_size, starpu_data_handle_t ∗data_handle, int ∗source, starpu_mpi_tag_t ∗data_tag, MPI_Comm ∗comm, starpu_tag_t tag)
- int starpu_mpi_datatype_register (starpu_data_handle_t handle, starpu_mpi_datatype_allocate_func_t allocate_datatype_func, starpu_mpi_datatype_free_func_t free_datatype_func)
- int starpu_mpi_interface_datatype_register (enum starpu_data_interface_id id, starpu_mpi_datatype_allocate_func_t allocate_datatype_func, starpu_mpi_datatype_free_func_t free_datatype_func)
- int starpu_mpi_datatype_node_register (starpu_data_handle_t handle, starpu_mpi_datatype_node_allocate_func_t allocate_datatype_func, starpu_mpi_datatype_free_func_t free_datatype_func)
- int starpu_mpi_interface_datatype_node_register (enum starpu_data_interface_id id, starpu_mpi_datatype_node_allocate_func_t allocate_datatype_func, starpu_mpi_datatype_free_func_t free_datatype_func)
- int starpu_mpi_datatype_unregister (starpu_data_handle_t handle)
- int starpu_mpi_interface_datatype_unregister (enum starpu_data_interface_id id)

59.25 starpu_mpi_ft.h File Reference

`#include <starpu.h>`

Typedefs

- typedef struct _starpu_mpi_checkpoint_template ∗ **starpu_mpi_checkpoint_template_t**

Functions

- int starpu_mpi_checkpoint_init (void)
- int starpu_mpi_checkpoint_shutdown (void)
- int starpu_mpi_checkpoint_template_register (starpu_mpi_checkpoint_template_t ∗cp_template, int cp_id, int cp_domain,...)
- int starpu_mpi_checkpoint_template_create (starpu_mpi_checkpoint_template_t ∗cp_template, int cp_id, int cp_domain)
- int starpu_mpi_checkpoint_template_add_entry (starpu_mpi_checkpoint_template_t ∗cp_template,...)
- int starpu_mpi_checkpoint_template_freeze (starpu_mpi_checkpoint_template_t ∗cp_template)
- int starpu_mpi_checkpoint_template_submit (starpu_mpi_checkpoint_template_t cp_template, int prio)
- int **starpu_mpi_checkpoint_template_print** (starpu_mpi_checkpoint_template_t cp_template)

59.26 starpu_mpi_lb.h File Reference

```
#include <starpu.h>
```

Data Structures

- struct starpu_mpi_lb_conf

Functions

- void starpu_mpi_lb_init (const char ∗lb_policy_name, struct starpu_mpi_lb_conf ∗)
- void **starpu_mpi_lb_shutdown** (void)

59.26.1 Function Documentation

59.26.1.1 starpu_mpi_lb_init()

```
void starpu_mpi_lb_init (
            const char * lb_policy_name,
            struct starpu_mpi_lb_conf *  )
```

Initialize the load balancer's environment with the load policy provided by the user

59.27 starpu_opencl.h File Reference

```
#include <starpu_config.h>
#include <CL/cl.h>
#include <assert.h>
```

Data Structures

- struct starpu_opencl_program

Macros

- #define **CL_TARGET_OPENCL_VERSION**

Functions

Writing OpenCL kernels

- void starpu_opencl_get_context (int devid, cl_context ∗context)
- void starpu_opencl_get_device (int devid, cl_device_id ∗device)
- void starpu_opencl_get_queue (int devid, cl_command_queue ∗queue)
- void starpu_opencl_get_current_context (cl_context ∗context)
- void starpu_opencl_get_current_queue (cl_command_queue ∗queue)
- int starpu_opencl_set_kernel_args (cl_int ∗err, cl_kernel ∗kernel,...)

Compiling OpenCL kernels

Source codes for OpenCL kernels can be stored in a file or in a string. StarPU provides functions to build the program executable for each available OpenCL device as a cl_program object. This program executable can then be loaded within a specific queue as explained in the next section. These are only helpers, Applications can also fill a starpu_opencl_program array by hand for more advanced use (e.g. different programs on the different OpenCL devices, for relocation purpose for instance).

  - void starpu_opencl_load_program_source (const char ∗source_file_name, char ∗located_file_name, char ∗located_dir_name, char ∗opencl_program_source)
  - void starpu_opencl_load_program_source_malloc (const char ∗source_file_name, char ∗∗located_file_name, char ∗∗located_dir_name, char ∗∗opencl_program_source)
  - int starpu_opencl_compile_opencl_from_file (const char ∗source_file_name, const char ∗build_options)
  - int starpu_opencl_compile_opencl_from_string (const char ∗opencl_program_source, const char ∗file_name, const char ∗build_options)
  - int starpu_opencl_load_binary_opencl (const char ∗kernel_id, struct starpu_opencl_program ∗opencl_programs)
  - int starpu_opencl_load_opencl_from_file (const char ∗source_file_name, struct starpu_opencl_program ∗opencl_programs, const char ∗build_options)
  - int starpu_opencl_load_opencl_from_string (const char ∗opencl_program_source, struct starpu_opencl_program ∗opencl_programs, const char ∗build_options)
  - int starpu_opencl_unload_opencl (struct starpu_opencl_program ∗opencl_programs)

Loading OpenCL kernels

  - int starpu_opencl_load_kernel (cl_kernel ∗kernel, cl_command_queue ∗queue, struct starpu_opencl_program ∗opencl_programs, const char ∗kernel_name, int devid)
  - int starpu_opencl_release_kernel (cl_kernel kernel)

OpenCL Statistics

  - int starpu_opencl_collect_stats (cl_event event)

OpenCL Utilities

- #define STARPU_OPENCL_DISPLAY_ERROR(status)
- #define STARPU_OPENCL_REPORT_ERROR(status)
- #define STARPU_OPENCL_REPORT_ERROR_WITH_MSG(msg, status)
- const char ∗ starpu_opencl_error_string (cl_int status)
- void starpu_opencl_display_error (const char ∗func, const char ∗file, int line, const char ∗msg, cl_int status)
- static __starpu_inline void starpu_opencl_report_error (const char ∗func, const char ∗file, int line, const char ∗msg, cl_int status)
- cl_int starpu_opencl_allocate_memory (int devid, cl_mem ∗addr, size_t size, cl_mem_flags flags)
- cl_int starpu_opencl_copy_ram_to_opencl (void ∗ptr, unsigned src_node, cl_mem buffer, unsigned dst_node, size_t size, size_t offset, cl_event ∗event, int ∗ret)
- cl_int starpu_opencl_copy_opencl_to_ram (cl_mem buffer, unsigned src_node, void ∗ptr, unsigned dst_node, size_t size, size_t offset, cl_event ∗event, int ∗ret)
- cl_int starpu_opencl_copy_opencl_to_opencl (cl_mem src, unsigned src_node, size_t src_offset, cl_mem dst, unsigned dst_node, size_t dst_offset, size_t size, cl_event ∗event, int ∗ret)
- cl_int starpu_opencl_copy_async_sync (uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, size_t dst_offset, unsigned dst_node, size_t size, cl_event ∗event)

59.28 starpu_openmp.h File Reference

`#include <starpu_config.h>`

Data Structures

- struct starpu_omp_lock_t
- struct starpu_omp_nest_lock_t
- struct starpu_omp_parallel_region_attr
- struct starpu_omp_task_region_attr

Macros

- #define **__STARPU_OMP_NOTHROW**

Enumerations

- enum starpu_omp_sched_value {
  starpu_omp_sched_undefined , starpu_omp_sched_static , starpu_omp_sched_dynamic , starpu_omp_sched_guided ,
  starpu_omp_sched_auto , starpu_omp_sched_runtime }
- enum starpu_omp_proc_bind_value {
  starpu_omp_proc_bind_undefined , starpu_omp_proc_bind_false , starpu_omp_proc_bind_true , starpu_omp_proc_bind_master ,
  starpu_omp_proc_bind_close , starpu_omp_proc_bind_spread }

Functions

Initialisation

- int starpu_omp_init (void) __STARPU_OMP_NOTHROW
- void starpu_omp_shutdown (void) __STARPU_OMP_NOTHROW

Parallel

- void starpu_omp_parallel_region (const struct starpu_omp_parallel_region_attr ∗attr) __STARPU_OMP_NOTHROW
- void starpu_omp_master (void(∗f)(void ∗arg), void ∗arg) __STARPU_OMP_NOTHROW
- int starpu_omp_master_inline (void) __STARPU_OMP_NOTHROW

Synchronization

- void starpu_omp_barrier (void) __STARPU_OMP_NOTHROW
- void starpu_omp_critical (void(∗f)(void ∗arg), void ∗arg, const char ∗name) __STARPU_OMP_NOTHROW
- void starpu_omp_critical_inline_begin (const char ∗name) __STARPU_OMP_NOTHROW
- void starpu_omp_critical_inline_end (const char ∗name) __STARPU_OMP_NOTHROW

Worksharing

- void starpu_omp_single (void(∗f)(void ∗arg), void ∗arg, int nowait) __STARPU_OMP_NOTHROW
- int starpu_omp_single_inline (void) __STARPU_OMP_NOTHROW
- void starpu_omp_single_copyprivate (void(∗f)(void ∗arg, void ∗data, unsigned long long data_size), void ∗arg, void ∗data, unsigned long long data_size) __STARPU_OMP_NOTHROW
- void ∗ starpu_omp_single_copyprivate_inline_begin (void ∗data) __STARPU_OMP_NOTHROW
- void starpu_omp_single_copyprivate_inline_end (void) __STARPU_OMP_NOTHROW
- void starpu_omp_for (void(∗f)(unsigned long long _first_i, unsigned long long _nb_i, void ∗arg), void ∗arg, unsigned long long nb_iterations, unsigned long long chunk, int schedule, int ordered, int nowait) __STARPU_OMP_NOTHROW
- int starpu_omp_for_inline_first (unsigned long long nb_iterations, unsigned long long chunk, int schedule, int ordered, unsigned long long ∗_first_i, unsigned long long ∗_nb_i) __STARPU_OMP_NOTHROW
- int starpu_omp_for_inline_next (unsigned long long nb_iterations, unsigned long long chunk, int schedule, int ordered, unsigned long long ∗_first_i, unsigned long long ∗_nb_i) __STARPU_OMP_NOTHROW
- void starpu_omp_for_alt (void(∗f)(unsigned long long _begin_i, unsigned long long _end_i, void ∗arg), void ∗arg, unsigned long long nb_iterations, unsigned long long chunk, int schedule, int ordered, int nowait) __STARPU_OMP_NOTHROW
- int starpu_omp_for_inline_first_alt (unsigned long long nb_iterations, unsigned long long chunk, int schedule, int ordered, unsigned long long ∗_begin_i, unsigned long long ∗_end_i) __STARPU_OMP_NOTHROW
- int starpu_omp_for_inline_next_alt (unsigned long long nb_iterations, unsigned long long chunk, int schedule, int ordered, unsigned long long ∗_begin_i, unsigned long long ∗_end_i) __STARPU_OMP_NOTHROW
- void starpu_omp_ordered (void(∗f)(void ∗arg), void ∗arg) __STARPU_OMP_NOTHROW
- void starpu_omp_ordered_inline_begin (void) __STARPU_OMP_NOTHROW
- void starpu_omp_ordered_inline_end (void) __STARPU_OMP_NOTHROW
- void starpu_omp_sections (unsigned long long nb_sections, void(∗∗section_f)(void ∗arg), void ∗∗section_arg, int nowait) __STARPU_OMP_NOTHROW
- void starpu_omp_sections_combined (unsigned long long nb_sections, void(∗section_f)(unsigned long long section_num, void ∗arg), void ∗section_arg, int nowait) __STARPU_OMP_NOTHROW

Task

- void starpu_omp_task_region (const struct starpu_omp_task_region_attr ∗attr) __STARPU_OMP_↩ NOTHROW
- void starpu_omp_taskwait (void) __STARPU_OMP_NOTHROW
- void starpu_omp_taskgroup (void(∗f)(void ∗arg), void ∗arg) __STARPU_OMP_NOTHROW
- void starpu_omp_taskgroup_inline_begin (void) __STARPU_OMP_NOTHROW
- void starpu_omp_taskgroup_inline_end (void) __STARPU_OMP_NOTHROW
- void **starpu_omp_taskloop_inline_begin** (struct starpu_omp_task_region_attr ∗attr) __STARPU_↩ OMP_NOTHROW
- void **starpu_omp_taskloop_inline_end** (const struct starpu_omp_task_region_attr ∗attr) __STARPU_↩ OMP_NOTHROW

API

- void starpu_omp_set_num_threads (int threads) __STARPU_OMP_NOTHROW
- int starpu_omp_get_num_threads (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_thread_num (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_max_threads (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_num_procs (void) __STARPU_OMP_NOTHROW
- int starpu_omp_in_parallel (void) __STARPU_OMP_NOTHROW
- void starpu_omp_set_dynamic (int dynamic_threads) __STARPU_OMP_NOTHROW
- int starpu_omp_get_dynamic (void) __STARPU_OMP_NOTHROW
- void starpu_omp_set_nested (int nested) __STARPU_OMP_NOTHROW
- int starpu_omp_get_nested (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_cancellation (void) __STARPU_OMP_NOTHROW
- void starpu_omp_set_schedule (enum starpu_omp_sched_value kind, int modifier) __STARPU_OMP_↩ NOTHROW
- void starpu_omp_get_schedule (enum starpu_omp_sched_value ∗kind, int ∗modifier) __STARPU_OMP↩ _NOTHROW
- int starpu_omp_get_thread_limit (void) __STARPU_OMP_NOTHROW
- void starpu_omp_set_max_active_levels (int max_levels) __STARPU_OMP_NOTHROW
- int starpu_omp_get_max_active_levels (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_level (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_ancestor_thread_num (int level) __STARPU_OMP_NOTHROW
- int starpu_omp_get_team_size (int level) __STARPU_OMP_NOTHROW
- int starpu_omp_get_active_level (void) __STARPU_OMP_NOTHROW
- int starpu_omp_in_final (void) __STARPU_OMP_NOTHROW
- enum starpu_omp_proc_bind_value starpu_omp_get_proc_bind (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_num_places (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_place_num_procs (int place_num) __STARPU_OMP_NOTHROW
- void starpu_omp_get_place_proc_ids (int place_num, int ∗ids) __STARPU_OMP_NOTHROW
- int starpu_omp_get_place_num (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_partition_num_places (void) __STARPU_OMP_NOTHROW
- void starpu_omp_get_partition_place_nums (int ∗place_nums) __STARPU_OMP_NOTHROW
- void starpu_omp_set_default_device (int device_num) __STARPU_OMP_NOTHROW
- int starpu_omp_get_default_device (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_num_devices (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_num_teams (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_team_num (void) __STARPU_OMP_NOTHROW
- int starpu_omp_is_initial_device (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_initial_device (void) __STARPU_OMP_NOTHROW
- int starpu_omp_get_max_task_priority (void) __STARPU_OMP_NOTHROW
- void starpu_omp_init_lock (starpu_omp_lock_t ∗lock) __STARPU_OMP_NOTHROW
- void starpu_omp_destroy_lock (starpu_omp_lock_t ∗lock) __STARPU_OMP_NOTHROW
- void starpu_omp_set_lock (starpu_omp_lock_t ∗lock) __STARPU_OMP_NOTHROW
- void starpu_omp_unset_lock (starpu_omp_lock_t ∗lock) __STARPU_OMP_NOTHROW
- int starpu_omp_test_lock (starpu_omp_lock_t ∗lock) __STARPU_OMP_NOTHROW
- void starpu_omp_init_nest_lock (starpu_omp_nest_lock_t ∗lock) __STARPU_OMP_NOTHROW
- void starpu_omp_destroy_nest_lock (starpu_omp_nest_lock_t ∗lock) __STARPU_OMP_NOTHROW
- void starpu_omp_set_nest_lock (starpu_omp_nest_lock_t ∗lock) __STARPU_OMP_NOTHROW
- void starpu_omp_unset_nest_lock (starpu_omp_nest_lock_t ∗lock) __STARPU_OMP_NOTHROW
- int starpu_omp_test_nest_lock (starpu_omp_nest_lock_t ∗lock) __STARPU_OMP_NOTHROW
- void starpu_omp_atomic_fallback_inline_begin (void) __STARPU_OMP_NOTHROW
- void starpu_omp_atomic_fallback_inline_end (void) __STARPU_OMP_NOTHROW
- double starpu_omp_get_wtime (void) __STARPU_OMP_NOTHROW

- double starpu_omp_get_wtick (void) __STARPU_OMP_NOTHROW
- void starpu_omp_vector_annotate (starpu_data_handle_t handle, uint32_t slice_base) __STARPU_OMP_NOTHROW
- struct starpu_arbiter ∗ starpu_omp_get_default_arbiter (void) __STARPU_OMP_NOTHROW
- void starpu_omp_handle_register (starpu_data_handle_t handle) __STARPU_OMP_NOTHROW
- void starpu_omp_handle_unregister (starpu_data_handle_t handle) __STARPU_OMP_NOTHROW
- starpu_data_handle_t starpu_omp_data_lookup (const void ∗ptr) __STARPU_OMP_NOTHROW

59.29 starpu_parallel_worker.h File Reference

```
#include <starpu_config.h>
#include <hwloc.h>
```

Macros

- #define STARPU_PARALLEL_WORKER_MIN_NB
- #define STARPU_PARALLEL_WORKER_MAX_NB
- #define STARPU_PARALLEL_WORKER_NB
- #define STARPU_PARALLEL_WORKER_PREFERE_MIN
- #define STARPU_PARALLEL_WORKER_KEEP_HOMOGENEOUS
- #define STARPU_PARALLEL_WORKER_POLICY_NAME
- #define STARPU_PARALLEL_WORKER_POLICY_STRUCT
- #define STARPU_PARALLEL_WORKER_CREATE_FUNC
- #define STARPU_PARALLEL_WORKER_CREATE_FUNC_ARG
- #define STARPU_PARALLEL_WORKER_TYPE
- #define STARPU_PARALLEL_WORKER_AWAKE_WORKERS
- #define STARPU_PARALLEL_WORKER_PARTITION_ONE
- #define STARPU_PARALLEL_WORKER_NEW
- #define STARPU_PARALLEL_WORKER_NCORES
- #define **starpu_parallel_worker_intel_openmp_mkl_prologue**
- #define STARPU_CLUSTER_MIN_NB
- #define STARPU_CLUSTER_MAX_NB
- #define STARPU_CLUSTER_NB
- #define STARPU_CLUSTER_PREFERE_MIN
- #define STARPU_CLUSTER_KEEP_HOMOGENEOUS
- #define STARPU_CLUSTER_POLICY_NAME
- #define STARPU_CLUSTER_POLICY_STRUCT
- #define STARPU_CLUSTER_CREATE_FUNC
- #define STARPU_CLUSTER_CREATE_FUNC_ARG
- #define STARPU_CLUSTER_TYPE
- #define STARPU_CLUSTER_AWAKE_WORKERS
- #define STARPU_CLUSTER_PARTITION_ONE
- #define STARPU_CLUSTER_NEW
- #define STARPU_CLUSTER_NCORES

Enumerations

- enum starpu_parallel_worker_types { STARPU_PARALLEL_WORKER_OPENMP , STARPU_PARALLEL_WORKER_INTEL_O , STARPU_PARALLEL_WORKER_GNU_OPENMP_MKL }
- enum starpu_cluster_types { STARPU_CLUSTER_OPENMP , STARPU_CLUSTER_INTEL_OPENMP_MKL , STARPU_CLUSTER_GNU_OPENMP_MKL }

Functions

- struct starpu_parallel_worker_config ∗ starpu_parallel_worker_init (hwloc_obj_type_t parallel_worker_level,...)
- int starpu_parallel_worker_shutdown (struct starpu_parallel_worker_config ∗parallel_workers)
- int starpu_parallel_worker_print (struct starpu_parallel_worker_config ∗parallel_workers)
- void starpu_parallel_worker_openmp_prologue (void ∗)
- void **starpu_parallel_worker_gnu_openmp_mkl_prologue** (void ∗)
- struct starpu_cluster_machine ∗ starpu_cluster_machine (hwloc_obj_type_t cluster_level,...)
- int starpu_uncluster_machine (struct starpu_cluster_machine ∗clusters)
- int starpu_cluster_print (struct starpu_cluster_machine ∗clusters)

59.30 starpu_perf_monitoring.h File Reference

`#include <starpu.h>`

Functions

Scope Related Routines

- int starpu_perf_counter_scope_name_to_id (const char ∗name)
- const char ∗ starpu_perf_counter_scope_id_to_name (enum starpu_perf_counter_scope scope)

Type Related Routines

- int starpu_perf_counter_type_name_to_id (const char ∗name)
- const char ∗ starpu_perf_counter_type_id_to_name (enum starpu_perf_counter_type type)

Counter Related Routines

- int starpu_perf_counter_nb (enum starpu_perf_counter_scope scope)
- int starpu_perf_counter_name_to_id (enum starpu_perf_counter_scope scope, const char ∗name)
- int starpu_perf_counter_nth_to_id (enum starpu_perf_counter_scope scope, int nth)
- const char ∗ starpu_perf_counter_id_to_name (int id)
- int starpu_perf_counter_get_type_id (int id)
- const char ∗ starpu_perf_counter_get_help_string (int id)

Listener Related Routines

- void starpu_perf_counter_list_avail (enum starpu_perf_counter_scope scope)
- void starpu_perf_counter_list_all_avail (void)
- struct starpu_perf_counter_set ∗ starpu_perf_counter_set_alloc (enum starpu_perf_counter_scope scope)
- void starpu_perf_counter_set_free (struct starpu_perf_counter_set ∗set)
- void starpu_perf_counter_set_enable_id (struct starpu_perf_counter_set ∗set, int id)
- void starpu_perf_counter_set_disable_id (struct starpu_perf_counter_set ∗set, int id)
- struct starpu_perf_counter_listener ∗ starpu_perf_counter_listener_init (struct starpu_perf_counter_set ∗set, void(∗callback)(struct starpu_perf_counter_listener ∗listener, struct starpu_perf_counter_sample ∗sample, void ∗context), void ∗user_arg)
- void starpu_perf_counter_listener_exit (struct starpu_perf_counter_listener ∗listener)
- void starpu_perf_counter_set_global_listener (struct starpu_perf_counter_listener ∗listener)
- void starpu_perf_counter_set_per_worker_listener (unsigned workerid, struct starpu_perf_counter_listener ∗listener)
- void starpu_perf_counter_set_all_per_worker_listeners (struct starpu_perf_counter_listener ∗listener)
- void starpu_perf_counter_set_per_codelet_listener (struct starpu_codelet ∗cl, struct starpu_perf_counter_listener ∗listener)
- void starpu_perf_counter_unset_global_listener (void)
- void starpu_perf_counter_unset_per_worker_listener (unsigned workerid)
- void starpu_perf_counter_unset_all_per_worker_listeners (void)
- void starpu_perf_counter_unset_per_codelet_listener (struct starpu_codelet ∗cl)

Sample Related Routines

- int32_t starpu_perf_counter_sample_get_int32_value (struct starpu_perf_counter_sample ∗sample, const int counter_id)
- int64_t starpu_perf_counter_sample_get_int64_value (struct starpu_perf_counter_sample ∗sample, const int counter_id)
- float starpu_perf_counter_sample_get_float_value (struct starpu_perf_counter_sample ∗sample, const int counter_id)
- double starpu_perf_counter_sample_get_double_value (struct starpu_perf_counter_sample ∗sample, const int counter_id)

API

- enum starpu_perf_counter_scope { starpu_perf_counter_scope_undefined , starpu_perf_counter_scope_global , starpu_perf_counter_scope_per_worker , starpu_perf_counter_scope_per_codelet }
- enum starpu_perf_counter_type {
  starpu_perf_counter_type_undefined , starpu_perf_counter_type_int32 , starpu_perf_counter_type_int64 ,
  starpu_perf_counter_type_float ,
  starpu_perf_counter_type_double }
- void starpu_perf_counter_collection_start (void)
- void starpu_perf_counter_collection_stop (void)

59.31 starpu_perf_steering.h File Reference

`#include <starpu.h>`

Enumerations

API

- enum starpu_perf_knob_scope { starpu_perf_knob_scope_undefined , starpu_perf_knob_scope_global , starpu_perf_knob_scope_per_worker , starpu_perf_knob_scope_per_scheduler }
- enum starpu_perf_knob_type {
  starpu_perf_knob_type_undefined , starpu_perf_knob_type_int32 , starpu_perf_knob_type_int64 ,
  starpu_perf_knob_type_float ,
  starpu_perf_knob_type_double }

Functions

Scope Related Routines

- int starpu_perf_knob_scope_name_to_id (const char ∗name)
- const char ∗ starpu_perf_knob_scope_id_to_name (enum starpu_perf_knob_scope scope)

Type Related Routines

- int starpu_perf_knob_type_name_to_id (const char ∗name)
- const char ∗ starpu_perf_knob_type_id_to_name (enum starpu_perf_knob_type type)

Performance Steering Knob Related Routines

- int starpu_perf_knob_nb (enum starpu_perf_knob_scope scope)
- int starpu_perf_knob_name_to_id (enum starpu_perf_knob_scope scope, const char ∗name)
- int starpu_perf_knob_nth_to_id (enum starpu_perf_knob_scope scope, int nth)
- const char ∗ starpu_perf_knob_id_to_name (int id)
- int starpu_perf_knob_get_type_id (int id)
- const char ∗ starpu_perf_knob_get_help_string (int id)
- void starpu_perf_knob_list_avail (enum starpu_perf_knob_scope scope)
- void starpu_perf_knob_list_all_avail (void)

- int32_t starpu_perf_knob_get_global_int32_value (const int knob_id)
- int64_t starpu_perf_knob_get_global_int64_value (const int knob_id)
- float starpu_perf_knob_get_global_float_value (const int knob_id)
- double starpu_perf_knob_get_global_double_value (const int knob_id)
- void starpu_perf_knob_set_global_int32_value (const int knob_id, int32_t new_value)
- void starpu_perf_knob_set_global_int64_value (const int knob_id, int64_t new_value)
- void starpu_perf_knob_set_global_float_value (const int knob_id, float new_value)
- void starpu_perf_knob_set_global_double_value (const int knob_id, double new_value)
- int32_t starpu_perf_knob_get_per_worker_int32_value (const int knob_id, unsigned workerid)
- int64_t starpu_perf_knob_get_per_worker_int64_value (const int knob_id, unsigned workerid)
- float starpu_perf_knob_get_per_worker_float_value (const int knob_id, unsigned workerid)
- double starpu_perf_knob_get_per_worker_double_value (const int knob_id, unsigned workerid)
- void starpu_perf_knob_set_per_worker_int32_value (const int knob_id, unsigned workerid, int32_t new_value)
- void starpu_perf_knob_set_per_worker_int64_value (const int knob_id, unsigned workerid, int64_t new_value)
- void starpu_perf_knob_set_per_worker_float_value (const int knob_id, unsigned workerid, float new_value)
- void starpu_perf_knob_set_per_worker_double_value (const int knob_id, unsigned workerid, double new_value)
- int32_t starpu_perf_knob_get_per_scheduler_int32_value (const int knob_id, const char ∗sched_policy_name)
- int64_t starpu_perf_knob_get_per_scheduler_int64_value (const int knob_id, const char ∗sched_policy_name)
- float starpu_perf_knob_get_per_scheduler_float_value (const int knob_id, const char ∗sched_policy_name)
- double starpu_perf_knob_get_per_scheduler_double_value (const int knob_id, const char ∗sched_policy_name)
- void starpu_perf_knob_set_per_scheduler_int32_value (const int knob_id, const char ∗sched_policy_name, int32_t new_value)
- void starpu_perf_knob_set_per_scheduler_int64_value (const int knob_id, const char ∗sched_policy_name, int64_t new_value)
- void starpu_perf_knob_set_per_scheduler_float_value (const int knob_id, const char ∗sched_policy_name, float new_value)
- void starpu_perf_knob_set_per_scheduler_double_value (const int knob_id, const char ∗sched_policy_name, double new_value)

59.32 starpu_perfmodel.h File Reference

```
#include <starpu.h>
#include <stdio.h>
```

Data Structures

- struct starpu_perfmodel_device
- struct starpu_perfmodel_arch
- struct starpu_perfmodel_history_entry
- struct starpu_perfmodel_history_list
- struct starpu_perfmodel_regression_model
- struct starpu_perfmodel_per_arch
- struct starpu_perfmodel

Macros

- #define **starpu_per_arch_perfmodel**

Typedefs

- typedef double(∗ **starpu_perfmodel_per_arch_cost_function**) (struct starpu_task ∗task, struct starpu_perfmodel_arch ∗arch, unsigned nimpl)
- typedef size_t(∗ **starpu_perfmodel_per_arch_size_base**) (struct starpu_task ∗task, struct starpu_perfmodel_arch ∗arch, unsigned nimpl)
- typedef struct _starpu_perfmodel_state ∗ **starpu_perfmodel_state_t**

Enumerations

- enum starpu_perfmodel_type {
  STARPU_PERFMODEL_INVALID , STARPU_PER_WORKER , STARPU_PER_ARCH , STARPU_COMMON ,
  STARPU_HISTORY_BASED , STARPU_REGRESSION_BASED , STARPU_NL_REGRESSION_BASED ,
  STARPU_MULTIPLE_REGRESSION_BASED }

Functions

- void starpu_perfmodel_init (struct starpu_perfmodel ∗model)
- int starpu_perfmodel_deinit (struct starpu_perfmodel ∗model)
- int starpu_energy_start (int workerid, enum starpu_worker_archtype archi)
- int starpu_energy_stop (struct starpu_perfmodel ∗model, struct starpu_task ∗task, unsigned nimpl, unsigned ntasks, int workerid, enum starpu_worker_archtype archi)
- int starpu_perfmodel_load_file (const char ∗filename, struct starpu_perfmodel ∗model)
- int starpu_perfmodel_load_symbol (const char ∗symbol, struct starpu_perfmodel ∗model)
- int starpu_perfmodel_unload_model (struct starpu_perfmodel ∗model)
- void starpu_save_history_based_model (struct starpu_perfmodel ∗model)
- void starpu_perfmodel_get_model_path (const char ∗symbol, char ∗path, size_t maxlen)
- void starpu_perfmodel_dump_xml (FILE ∗output, struct starpu_perfmodel ∗model)
- void starpu_perfmodel_free_sampling (void)
- struct starpu_perfmodel_arch ∗ starpu_worker_get_perf_archtype (int workerid, unsigned sched_ctx_id)
- int **starpu_perfmodel_get_narch_combs** (void)
- int **starpu_perfmodel_arch_comb_add** (int ndevices, struct starpu_perfmodel_device ∗devices)
- int **starpu_perfmodel_arch_comb_get** (int ndevices, struct starpu_perfmodel_device ∗devices)
- struct starpu_perfmodel_arch ∗ **starpu_perfmodel_arch_comb_fetch** (int comb)
- struct starpu_perfmodel_per_arch ∗ **starpu_perfmodel_get_model_per_arch** (struct starpu_perfmodel ∗model, struct starpu_perfmodel_arch ∗arch, unsigned impl)
- struct starpu_perfmodel_per_arch ∗ **starpu_perfmodel_get_model_per_devices** (struct starpu_perfmodel ∗model, int impl,...)
- int **starpu_perfmodel_set_per_devices_cost_function** (struct starpu_perfmodel ∗model, int impl, starpu_perfmodel_per_arch_cost_function func,...)
- int **starpu_perfmodel_set_per_devices_size_base** (struct starpu_perfmodel ∗model, int impl, starpu_perfmodel_per_arch_size_base func,...)
- void starpu_perfmodel_debugfilepath (struct starpu_perfmodel ∗model, struct starpu_perfmodel_arch ∗arch, char ∗path, size_t maxlen, unsigned nimpl)
- const char ∗ **starpu_perfmodel_get_archtype_name** (enum starpu_worker_archtype archtype)
- void starpu_perfmodel_get_arch_name (struct starpu_perfmodel_arch ∗arch, char ∗archname, size_t maxlen, unsigned nimpl)
- double starpu_perfmodel_history_based_expected_perf (struct starpu_perfmodel ∗model, struct starpu_perfmodel_arch ∗arch, uint32_t footprint)
- void starpu_perfmodel_initialize (void)
- int starpu_perfmodel_list (FILE ∗output)
- void **starpu_perfmodel_print** (struct starpu_perfmodel ∗model, struct starpu_perfmodel_arch ∗arch, unsigned nimpl, char ∗parameter, uint32_t ∗footprint, FILE ∗output)
- int **starpu_perfmodel_print_all** (struct starpu_perfmodel ∗model, char ∗arch, char ∗parameter, uint32_t ∗footprint, FILE ∗output)
- int **starpu_perfmodel_print_estimations** (struct starpu_perfmodel ∗model, uint32_t footprint, FILE ∗output)

- int **starpu_perfmodel_list_combs** (FILE ∗output, struct starpu_perfmodel ∗model)
- void starpu_perfmodel_update_history (struct starpu_perfmodel ∗model, struct starpu_task ∗task, struct starpu_perfmodel_arch ∗arch, unsigned cpuid, unsigned nimpl, double measured)
- void starpu_perfmodel_update_history_n (struct starpu_perfmodel ∗model, struct starpu_task ∗task, struct starpu_perfmodel_arch ∗arch, unsigned cpuid, unsigned nimpl, double average_measured, unsigned number)
- void starpu_perfmodel_directory (FILE ∗output)
- void starpu_bus_print_bandwidth (FILE ∗f)
- void starpu_bus_print_affinity (FILE ∗f)
- void starpu_bus_print_filenames (FILE ∗f)
- double starpu_transfer_bandwidth (unsigned src_node, unsigned dst_node)
- double starpu_transfer_latency (unsigned src_node, unsigned dst_node)
- double starpu_transfer_predict (unsigned src_node, unsigned dst_node, size_t size)

Variables

- struct starpu_perfmodel starpu_perfmodel_nop

59.33 starpu_profiling.h File Reference

```
#include <starpu.h>
#include <errno.h>
#include <time.h>
#include <starpu_config.h>
```

Data Structures

- struct starpu_profiling_task_info
- struct starpu_profiling_worker_info
- struct starpu_profiling_bus_info

Macros

- #define STARPU_PROFILING_DISABLE
- #define STARPU_PROFILING_ENABLE
- #define **STARPU_NS_PER_S**
- #define **starpu_timespec_cmp**(a, b, CMP)

Functions

- void starpu_profiling_init (void)
- void starpu_profiling_set_id (int new_id)
- int starpu_profiling_status_set (int status)
- int starpu_profiling_status_get (void)
- int starpu_profiling_worker_get_info (int workerid, struct starpu_profiling_worker_info ∗worker_info)
- int starpu_bus_get_count (void)
- int starpu_bus_get_id (int src, int dst)
- int starpu_bus_get_src (int busid)
- int starpu_bus_get_dst (int busid)
- void starpu_bus_set_direct (int busid, int direct)
- int starpu_bus_get_direct (int busid)
- void starpu_bus_set_ngpus (int busid, int ngpus)
- int starpu_bus_get_ngpus (int busid)
- int starpu_bus_get_profiling_info (int busid, struct starpu_profiling_bus_info ∗bus_info)

- static __starpu_inline void **starpu_timespec_clear** (struct timespec ∗tsp)
- static __starpu_inline void **starpu_timespec_add** (struct timespec ∗a, struct timespec ∗b, struct timespec ∗result)
- static __starpu_inline void **starpu_timespec_accumulate** (struct timespec ∗result, struct timespec ∗a)
- static __starpu_inline void **starpu_timespec_sub** (const struct timespec ∗a, const struct timespec ∗b, struct timespec ∗result)
- double starpu_timing_timespec_delay_us (struct timespec ∗start, struct timespec ∗end)
- double starpu_timing_timespec_to_us (struct timespec ∗ts)
- void starpu_profiling_bus_helper_display_summary (void)
- void starpu_profiling_worker_helper_display_summary (void)
- void starpu_data_display_memory_stats (void)

59.34 starpu_profiling_tool.h File Reference

`#include <starpu.h>`

Data Structures

- struct starpu_prof_tool_info
- union starpu_prof_tool_event_info
- struct starpu_prof_tool_api_info

Typedefs

- typedef void(∗ **starpu_prof_tool_cb_func**) (struct starpu_prof_tool_info ∗, union starpu_prof_tool_event_info ∗, struct starpu_prof_tool_api_info ∗)
- typedef void(∗ starpu_prof_tool_entry_register_func) (enum starpu_prof_tool_event event_type, starpu_prof_tool_cb_func cb, enum starpu_prof_tool_command info)
- typedef void(∗ starpu_prof_tool_entry_func) (starpu_prof_tool_entry_register_func reg, starpu_prof_tool_entry_register_func unreg)

Enumerations

- enum starpu_prof_tool_event {
starpu_prof_tool_event_none , **starpu_prof_tool_event_init** , **starpu_prof_tool_event_terminate** , **starpu_prof_tool_event_init_begin** ,
starpu_prof_tool_event_init_end , **starpu_prof_tool_event_driver_init** , **starpu_prof_tool_event_driver_deinit** , **starpu_prof_tool_event_driver_init_start** ,
starpu_prof_tool_event_driver_init_end , **starpu_prof_tool_event_start_cpu_exec** , **starpu_prof_tool_event_end_cpu_exec** , **starpu_prof_tool_event_start_gpu_exec** ,
starpu_prof_tool_event_end_gpu_exec , **starpu_prof_tool_event_start_transfer** , **starpu_prof_tool_event_end_transfer** , **starpu_prof_tool_event_user_start** ,
starpu_prof_tool_event_user_end }
- enum starpu_prof_tool_driver_type { **starpu_prof_tool_driver_cpu** , **starpu_prof_tool_driver_gpu** , **starpu_prof_tool_driver_hip** , **starpu_prof_tool_driver_ocl** }
- enum starpu_prof_tool_command { **starpu_prof_tool_command_reg** , **starpu_prof_tool_command_toggle** , **starpu_prof_tool_command_toggle_per_thread** }

59.35 starpu_rand.h File Reference

```
#include <stdlib.h>
#include <starpu_config.h>
```

Macros

- #define **starpu_seed**(seed)
- #define **starpu_srand48**(seed)
- #define **starpu_drand48**()
- #define **starpu_lrand48**()
- #define **starpu_erand48**(xsubi)
- #define **starpu_srand48_r**(seed, buffer)
- #define **starpu_erand48_r**(xsubi, buffer, result)

Typedefs

- typedef int **starpu_drand48_data**

59.36 starpu_sched_component.h File Reference

```
#include <starpu.h>
#include <hwloc.h>
```

Data Structures

- struct starpu_sched_component
- struct starpu_sched_tree
- struct starpu_sched_component_fifo_data
- struct starpu_sched_component_prio_data
- struct starpu_sched_component_mct_data
- struct starpu_sched_component_heteroprio_data
- struct starpu_sched_component_perfmodel_select_data
- struct starpu_sched_component_specs

Macros

- #define STARPU_SCHED_COMPONENT_IS_HOMOGENEOUS(component)
- #define STARPU_SCHED_COMPONENT_IS_SINGLE_MEMORY_NODE(component)
- #define **STARPU_COMPONENT_MUTEX_LOCK**(m)
- #define **STARPU_COMPONENT_MUTEX_TRYLOCK**(m)
- #define **STARPU_COMPONENT_MUTEX_UNLOCK**(m)

Enumerations

- enum starpu_sched_component_properties { STARPU_SCHED_COMPONENT_HOMOGENEOUS , STARPU_SCHED_COMPONENT_SINGLE_MEMORY_NODE }

Functions

Scheduling Tree API

- struct starpu_sched_tree ∗ starpu_sched_tree_create (unsigned sched_ctx_id) STARPU_ATTRIBUTE_MALLOC
- void starpu_sched_tree_destroy (struct starpu_sched_tree ∗tree)
- void starpu_sched_tree_deinitialize (unsigned sched_ctx_id)
- struct starpu_sched_tree ∗ starpu_sched_tree_get (unsigned sched_ctx_id)
- void starpu_sched_tree_update_workers (struct starpu_sched_tree ∗t)
- void starpu_sched_tree_update_workers_in_ctx (struct starpu_sched_tree ∗t)
- int starpu_sched_tree_push_task (struct starpu_task ∗task)
- struct starpu_task ∗ starpu_sched_tree_pop_task (unsigned sched_ctx)
- int starpu_sched_component_push_task (struct starpu_sched_component ∗from, struct starpu_sched_component ∗to, struct starpu_task ∗task)

- struct starpu_task ∗ starpu_sched_component_pull_task (struct starpu_sched_component ∗from, struct starpu_sched_component ∗to)
- struct starpu_task ∗ **starpu_sched_component_pump_to** (struct starpu_sched_component ∗component, struct starpu_sched_component ∗to, int ∗success)
- struct starpu_task ∗ **starpu_sched_component_pump_downstream** (struct starpu_sched_component ∗component, int ∗success)
- int **starpu_sched_component_send_can_push_to_parents** (struct starpu_sched_component ∗component)
- void starpu_sched_tree_add_workers (unsigned sched_ctx_id, int ∗workerids, unsigned nworkers)
- void starpu_sched_tree_remove_workers (unsigned sched_ctx_id, int ∗workerids, unsigned nworkers)
- void starpu_sched_tree_do_schedule (unsigned sched_ctx_id)
- void starpu_sched_component_connect (struct starpu_sched_component ∗parent, struct starpu_sched_component ∗child)

Worker Component API

- struct starpu_sched_component ∗ starpu_sched_component_worker_get (unsigned sched_ctx, int workerid)
- struct starpu_sched_component ∗ **starpu_sched_component_worker_new** (unsigned sched_ctx, int workerid)
- struct starpu_sched_component ∗ starpu_sched_component_parallel_worker_create (struct starpu_sched_tree ∗tree, unsigned nworkers, unsigned ∗workers)
- int starpu_sched_component_worker_get_workerid (struct starpu_sched_component ∗worker_component)
- int starpu_sched_component_is_worker (struct starpu_sched_component ∗component)
- int starpu_sched_component_is_simple_worker (struct starpu_sched_component ∗component)
- int starpu_sched_component_is_combined_worker (struct starpu_sched_component ∗component)
- void starpu_sched_component_worker_pre_exec_hook (struct starpu_task ∗task, unsigned sched_ctx_id)
- void starpu_sched_component_worker_post_exec_hook (struct starpu_task ∗task, unsigned sched_ctx_id)

Flow-control Fifo Component API

These can be used as methods of components. Note: they are not to be called directly, one should really call the methods of the components.

- struct starpu_task ∗ starpu_sched_component_parents_pull_task (struct starpu_sched_component ∗component, struct starpu_sched_component ∗to)
- int starpu_sched_component_can_push (struct starpu_sched_component ∗component, struct starpu_sched_component ∗to)
- int starpu_sched_component_can_pull (struct starpu_sched_component ∗component)
- int starpu_sched_component_can_pull_all (struct starpu_sched_component ∗component)
- double starpu_sched_component_estimated_load (struct starpu_sched_component ∗component)
- double starpu_sched_component_estimated_end_min (struct starpu_sched_component ∗component)
- double starpu_sched_component_estimated_end_min_add (struct starpu_sched_component ∗component, double exp_len)
- double starpu_sched_component_estimated_end_average (struct starpu_sched_component ∗component)
- struct starpu_sched_component ∗ starpu_sched_component_fifo_create (struct starpu_sched_tree ∗tree, struct starpu_sched_component_fifo_data ∗fifo_data) STARPU_ATTRIBUTE_MALLOC
- int starpu_sched_component_is_fifo (struct starpu_sched_component ∗component)

Flow-control Prio Component API

- struct starpu_sched_component ∗ **starpu_sched_component_prio_create** (struct starpu_sched_tree ∗tree, struct starpu_sched_component_prio_data ∗prio_data) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_prio** (struct starpu_sched_component ∗component)

Resource-mapping Work-Stealing Component API

- struct starpu_sched_component ∗ starpu_sched_component_work_stealing_create (struct starpu_sched_tree ∗tree, void ∗arg) STARPU_ATTRIBUTE_MALLOC
- int starpu_sched_component_is_work_stealing (struct starpu_sched_component ∗component)
- int starpu_sched_tree_work_stealing_push_task (struct starpu_task ∗task)

Resource-mapping Random Component API

- struct starpu_sched_component ∗ starpu_sched_component_random_create (struct starpu_sched_tree ∗tree, void ∗arg) STARPU_ATTRIBUTE_MALLOC
- int starpu_sched_component_is_random (struct starpu_sched_component ∗)

Resource-mapping Eager Component API

- struct starpu_sched_component ∗ **starpu_sched_component_eager_create** (struct starpu_sched_tree ∗tree, void ∗arg) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_eager** (struct starpu_sched_component ∗)

Resource-mapping Eager Prio Component API

- struct starpu_sched_component ∗ **starpu_sched_component_eager_prio_create** (struct starpu_sched_tree ∗tree, void ∗arg) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_eager_prio** (struct starpu_sched_component ∗)

Resource-mapping Eager-Calibration Component API

- struct starpu_sched_component ∗ **starpu_sched_component_eager_calibration_create** (struct starpu_sched_tree ∗tree, void ∗arg) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_eager_calibration** (struct starpu_sched_component ∗)

Resource-mapping MCT Component API

- struct starpu_sched_component ∗ starpu_sched_component_mct_create (struct starpu_sched_tree ∗tree, struct starpu_sched_component_mct_data ∗mct_data) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_mct** (struct starpu_sched_component ∗component)

Resource-mapping Heft Component API

- struct starpu_sched_component ∗ **starpu_sched_component_heft_create** (struct starpu_sched_tree ∗tree, struct starpu_sched_component_mct_data ∗mct_data) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_heft** (struct starpu_sched_component ∗component)

Resource-mapping Heteroprio Component API

- struct starpu_sched_component ∗ **starpu_sched_component_heteroprio_create** (struct starpu_sched_tree ∗tree, struct starpu_sched_component_heteroprio_data ∗params) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_heteroprio** (struct starpu_sched_component ∗component)

Special-purpose Best_Implementation Component API

- struct starpu_sched_component ∗ starpu_sched_component_best_implementation_create (struct starpu_sched_tree ∗tree, void ∗arg) STARPU_ATTRIBUTE_MALLOC

Special-purpose Perfmodel_Select Component API

- struct starpu_sched_component ∗ **starpu_sched_component_perfmodel_select_create** (struct starpu_sched_tree ∗tree, struct starpu_sched_component_perfmodel_select_data ∗perfmodel_select_data) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_perfmodel_select** (struct starpu_sched_component ∗component)

Staged pull Component API

- struct starpu_sched_component ∗ **starpu_sched_component_stage_create** (struct starpu_sched_tree ∗tree, void ∗arg) STARPU_ATTRIBUTE_MALLOC
- int **starpu_sched_component_is_stage** (struct starpu_sched_component ∗component)

User-choice push Component API

- struct starpu_sched_component ∗ **starpu_sched_component_userchoice_create** (struct starpu_sched_tree ∗tree, void ∗arg) STARPU_ATTRIBUTE_MALLOC

- int **starpu_sched_component_is_userchoice** (struct starpu_sched_component ∗component)

Recipe Component API

- struct starpu_sched_component_composed_recipe ∗ starpu_sched_component_composed_recipe_create (void) STARPU_ATTRIBUTE_MALLOC
- struct starpu_sched_component_composed_recipe ∗ starpu_sched_component_composed_recipe_create_singleton (struct starpu_sched_component ∗(∗create_component)(struct starpu_sched_tree ∗tree, void ∗arg), void ∗arg) STARPU_ATTRIBUTE_MALLOC
- void starpu_sched_component_composed_recipe_add (struct starpu_sched_component_composed_↩ recipe ∗recipe, struct starpu_sched_component ∗(∗create_component)(struct starpu_sched_tree ∗tree, void ∗arg), void ∗arg)
- void starpu_sched_component_composed_recipe_destroy (struct starpu_sched_component_composed↩ _recipe ∗)
- struct starpu_sched_component ∗ starpu_sched_component_composed_component_create (struct starpu_sched_tree ∗tree, struct starpu_sched_component_composed_recipe ∗recipe) STARPU_ATTRIBUTE_MALLOC
- struct starpu_sched_tree ∗ starpu_sched_component_make_scheduler (unsigned sched_ctx_id, struct starpu_sched_component_specs s)

Basic API

- #define **STARPU_SCHED_SIMPLE_DECIDE_MASK**
- #define STARPU_SCHED_SIMPLE_DECIDE_WORKERS
- #define STARPU_SCHED_SIMPLE_DECIDE_MEMNODES
- #define STARPU_SCHED_SIMPLE_DECIDE_ARCHS
- #define STARPU_SCHED_SIMPLE_DECIDE_ALWAYS
- #define STARPU_SCHED_SIMPLE_PERFMODEL
- #define STARPU_SCHED_SIMPLE_IMPL
- #define STARPU_SCHED_SIMPLE_FIFO_ABOVE
- #define STARPU_SCHED_SIMPLE_FIFO_ABOVE_PRIO
- #define STARPU_SCHED_SIMPLE_FIFOS_BELOW
- #define STARPU_SCHED_SIMPLE_FIFOS_BELOW_PRIO
- #define STARPU_SCHED_SIMPLE_FIFOS_BELOW_READY
- #define STARPU_SCHED_SIMPLE_WS_BELOW
- #define STARPU_SCHED_SIMPLE_COMBINED_WORKERS
- #define STARPU_SCHED_SIMPLE_FIFOS_BELOW_EXP
- #define STARPU_SCHED_SIMPLE_PRE_DECISION
- void starpu_sched_component_initialize_simple_scheduler (starpu_sched_component_create_t create_↩ decision_component, void ∗data, unsigned flags, unsigned sched_ctx_id)
- void starpu_sched_component_initialize_simple_schedulers (unsigned sched_ctx_id, unsigned ndecisions,...)

Generic Scheduling Component API

- typedef struct starpu_sched_component ∗(∗ **starpu_sched_component_create_t**) (struct starpu_sched_tree ∗tree, void ∗data)
- struct starpu_sched_component ∗ starpu_sched_component_create (struct starpu_sched_tree ∗tree, const char ∗name) STARPU_ATTRIBUTE_MALLOC
- void starpu_sched_component_destroy (struct starpu_sched_component ∗component)
- void starpu_sched_component_destroy_rec (struct starpu_sched_component ∗component)
- void **starpu_sched_component_add_child** (struct starpu_sched_component ∗component, struct starpu_sched_component ∗child)
- int starpu_sched_component_can_execute_task (struct starpu_sched_component ∗component, struct starpu_task ∗task)
- int starpu_sched_component_execute_preds (struct starpu_sched_component ∗component, struct starpu_task ∗task, double ∗length)
- double starpu_sched_component_transfer_length (struct starpu_sched_component ∗component, struct starpu_task ∗task)
- void **starpu_sched_component_prefetch_on_node** (struct starpu_sched_component ∗component, struct starpu_task ∗task)

59.37 starpu_sched_ctx.h File Reference

```
#include <starpu.h>
```

Functions

Scheduling Context Worker Collection

- struct starpu_worker_collection ∗ starpu_sched_ctx_create_worker_collection (unsigned sched_ctx_id, enum starpu_worker_collection_type type) STARPU_ATTRIBUTE_MALLOC
- void starpu_sched_ctx_delete_worker_collection (unsigned sched_ctx_id)
- struct starpu_worker_collection ∗ starpu_sched_ctx_get_worker_collection (unsigned sched_ctx_id)

Scheduling Contexts Basic API

- #define STARPU_SCHED_CTX_POLICY_NAME
- #define STARPU_SCHED_CTX_POLICY_STRUCT
- #define STARPU_SCHED_CTX_POLICY_MIN_PRIO
- #define STARPU_SCHED_CTX_POLICY_MAX_PRIO
- #define **STARPU_SCHED_CTX_HIERARCHY_LEVEL**
- #define **STARPU_SCHED_CTX_NESTED**
- #define STARPU_SCHED_CTX_AWAKE_WORKERS
- #define STARPU_SCHED_CTX_POLICY_INIT
- #define STARPU_SCHED_CTX_USER_DATA
- #define STARPU_SCHED_CTX_CUDA_NSMS
- #define STARPU_SCHED_CTX_SUB_CTXS
- void(∗)(unsigned) starpu_sched_ctx_get_sched_policy_callback (unsigned sched_ctx_id)
- unsigned starpu_sched_ctx_create (int ∗workerids_ctx, int nworkers_ctx, const char ∗sched_ctx_name,...)
- unsigned starpu_sched_ctx_create_inside_interval (const char ∗policy_name, const char ∗sched_ctx_name, int min_ncpus, int max_ncpus, int min_ngpus, int max_ngpus, unsigned allow_overlap)
- void starpu_sched_ctx_register_close_callback (unsigned sched_ctx_id, void(∗close_callback)(unsigned sched_ctx_id, void ∗args), void ∗args)
- void starpu_sched_ctx_add_workers (int ∗workerids_ctx, unsigned nworkers_ctx, unsigned sched_ctx_id)
- void starpu_sched_ctx_remove_workers (int ∗workerids_ctx, unsigned nworkers_ctx, unsigned sched_ctx_id)
- void starpu_sched_ctx_display_workers (unsigned sched_ctx_id, FILE ∗f)
- void starpu_sched_ctx_delete (unsigned sched_ctx_id)
- void starpu_sched_ctx_set_inheritor (unsigned sched_ctx_id, unsigned inheritor)
- unsigned **starpu_sched_ctx_get_inheritor** (unsigned sched_ctx_id)
- unsigned **starpu_sched_ctx_get_hierarchy_level** (unsigned sched_ctx_id)
- void starpu_sched_ctx_set_context (unsigned ∗sched_ctx_id)
- unsigned starpu_sched_ctx_get_context (void)
- void starpu_sched_ctx_stop_task_submission (void)
- void starpu_sched_ctx_finished_submit (unsigned sched_ctx_id)
- unsigned starpu_sched_ctx_get_workers_list (unsigned sched_ctx_id, int ∗∗workerids)
- unsigned starpu_sched_ctx_get_workers_list_raw (unsigned sched_ctx_id, int ∗∗workerids)
- unsigned starpu_sched_ctx_get_nworkers (unsigned sched_ctx_id)
- unsigned starpu_sched_ctx_get_nshared_workers (unsigned sched_ctx_id, unsigned sched_ctx_id2)
- unsigned starpu_sched_ctx_contains_worker (int workerid, unsigned sched_ctx_id)
- unsigned **starpu_sched_ctx_contains_type_of_worker** (enum starpu_worker_archtype arch, unsigned sched_ctx_id)
- unsigned starpu_sched_ctx_worker_get_id (unsigned sched_ctx_id)
- unsigned **starpu_sched_ctx_get_ctx_for_task** (struct starpu_task ∗task)
- unsigned **starpu_worker_get_sched_ctx_id_stream** (unsigned stream_workerid)
- unsigned starpu_sched_ctx_overlapping_ctxs_on_worker (int workerid)

- void ∗ starpu_sched_ctx_get_user_data (unsigned sched_ctx_id)
- void **starpu_sched_ctx_set_user_data** (unsigned sched_ctx_id, void ∗user_data)
- void starpu_sched_ctx_set_policy_data (unsigned sched_ctx_id, void ∗policy_data)
- void ∗ starpu_sched_ctx_get_policy_data (unsigned sched_ctx_id)
- struct starpu_sched_policy ∗ **starpu_sched_ctx_get_sched_policy** (unsigned sched_ctx_id)
- void ∗ starpu_sched_ctx_exec_parallel_code (void ∗(∗func)(void ∗), void ∗param, unsigned sched_ctx_id)
- int **starpu_sched_ctx_get_nready_tasks** (unsigned sched_ctx_id)
- double **starpu_sched_ctx_get_nready_flops** (unsigned sched_ctx_id)
- void **starpu_sched_ctx_list_task_counters_increment** (unsigned sched_ctx_id, int workerid)
- void **starpu_sched_ctx_list_task_counters_decrement** (unsigned sched_ctx_id, int workerid)
- void **starpu_sched_ctx_list_task_counters_reset** (unsigned sched_ctx_id, int workerid)
- void **starpu_sched_ctx_list_task_counters_increment_all_ctx_locked** (struct starpu_task ∗task, unsigned sched_ctx_id)
- void **starpu_sched_ctx_list_task_counters_decrement_all_ctx_locked** (struct starpu_task ∗task, unsigned sched_ctx_id)
- void **starpu_sched_ctx_list_task_counters_reset_all** (struct starpu_task ∗task, unsigned sched_ctx_id)
- void **starpu_sched_ctx_set_priority** (int ∗workers, int nworkers, unsigned sched_ctx_id, unsigned priority)
- unsigned **starpu_sched_ctx_get_priority** (int worker, unsigned sched_ctx_id)
- void **starpu_sched_ctx_get_available_cpuids** (unsigned sched_ctx_id, int ∗∗cpuids, int ∗ncpuids)
- void **starpu_sched_ctx_bind_current_thread_to_cpuid** (unsigned cpuid)
- int **starpu_sched_ctx_book_workers_for_task** (unsigned sched_ctx_id, int ∗workerids, int nworkers)
- void **starpu_sched_ctx_unbook_workers_for_task** (unsigned sched_ctx_id, int master)
- unsigned starpu_sched_ctx_worker_is_master_for_child_ctx (int workerid, unsigned sched_ctx_id)
- unsigned starpu_sched_ctx_master_get_context (int masterid)
- void **starpu_sched_ctx_revert_task_counters_ctx_locked** (unsigned sched_ctx_id, double flops)
- void **starpu_sched_ctx_move_task_to_ctx_locked** (struct starpu_task ∗task, unsigned sched_ctx, unsigned with_repush)
- int **starpu_sched_ctx_get_worker_rank** (unsigned sched_ctx_id)
- unsigned **starpu_sched_ctx_has_starpu_scheduler** (unsigned sched_ctx_id, unsigned ∗awake_workers)
- int **starpu_sched_ctx_get_stream_worker** (unsigned sub_ctx)
- int **starpu_sched_ctx_get_nsms** (unsigned sched_ctx)
- void **starpu_sched_ctx_get_sms_interval** (int stream_workerid, int ∗start, int ∗end)

Scheduling Context Priorities

- #define STARPU_MIN_PRIO
- #define STARPU_MAX_PRIO
- #define STARPU_DEFAULT_PRIO
- int starpu_sched_ctx_get_min_priority (unsigned sched_ctx_id)
- int starpu_sched_ctx_get_max_priority (unsigned sched_ctx_id)
- int starpu_sched_ctx_set_min_priority (unsigned sched_ctx_id, int min_prio)
- int starpu_sched_ctx_set_max_priority (unsigned sched_ctx_id, int max_prio)
- int **starpu_sched_ctx_min_priority_is_set** (unsigned sched_ctx_id)
- int **starpu_sched_ctx_max_priority_is_set** (unsigned sched_ctx_id)

59.38 starpu_sched_ctx_hypervisor.h File Reference

Data Structures

- struct starpu_sched_ctx_performance_counters

Functions

Scheduling Context Link with Hypervisor

- void starpu_sched_ctx_set_perf_counters (unsigned sched_ctx_id, void ∗perf_counters)
- void starpu_sched_ctx_call_pushed_task_cb (int workerid, unsigned sched_ctx_id)
- void starpu_sched_ctx_notify_hypervisor_exists (void)
- unsigned starpu_sched_ctx_check_if_hypervisor_exists (void)
- void **starpu_sched_ctx_update_start_resizing_sample** (unsigned sched_ctx_id, double start_sample)

59.38.1 Function Documentation

59.38.1.1 starpu_sched_ctx_set_perf_counters()

```
void starpu_sched_ctx_set_perf_counters (
            unsigned sched_ctx_id,
            void * perf_counters )
```

Indicate to starpu the pointer to the performance counter

59.38.1.2 starpu_sched_ctx_call_pushed_task_cb()

```
void starpu_sched_ctx_call_pushed_task_cb (
            int workerid,
            unsigned sched_ctx_id )
```

Callback that lets the scheduling policy tell the hypervisor that a task was pushed on a worker

59.38.1.3 starpu_sched_ctx_notify_hypervisor_exists()

```
void starpu_sched_ctx_notify_hypervisor_exists (
            void )
```

Allow the hypervisor to let starpu know it's initialised

59.38.1.4 starpu_sched_ctx_check_if_hypervisor_exists()

```
unsigned starpu_sched_ctx_check_if_hypervisor_exists (
            void )
```

Ask starpu if it is informed if the hypervisor is initialised

59.39 starpu_scheduler.h File Reference

```
#include <starpu.h>
```

Data Structures

- struct starpu_sched_policy

Typedefs

- typedef void(∗ **starpu_notify_ready_soon_func**) (void ∗data, struct starpu_task ∗task, double delay)

Functions

- struct starpu_sched_policy ∗∗ starpu_sched_get_predefined_policies (void)
- struct starpu_sched_policy ∗ starpu_get_sched_lib_policy (const char ∗name)
- struct starpu_sched_policy ∗∗ starpu_get_sched_lib_policies (void)
- struct starpu_sched_policy ∗ starpu_sched_get_sched_policy_in_ctx (unsigned sched_ctx_id)

- struct starpu_sched_policy ∗ starpu_sched_get_sched_policy (void)
- void starpu_worker_get_sched_condition (int workerid, starpu_pthread_mutex_t ∗∗sched_mutex, starpu_pthread_cond_t ∗∗sched_cond)
- unsigned long starpu_task_get_job_id (struct starpu_task ∗task)
- int starpu_sched_get_min_priority (void)
- int starpu_sched_get_max_priority (void)
- int starpu_sched_set_min_priority (int min_prio)
- int starpu_sched_set_max_priority (int max_prio)
- int starpu_worker_can_execute_task (unsigned workerid, struct starpu_task ∗task, unsigned nimpl)
- int starpu_worker_can_execute_task_impl (unsigned workerid, struct starpu_task ∗task, unsigned ∗impl_mask)
- int starpu_worker_can_execute_task_first_impl (unsigned workerid, struct starpu_task ∗task, unsigned ∗nimpl)
- int starpu_push_local_task (int workerid, struct starpu_task ∗task, int back)
- int starpu_push_task_end (struct starpu_task ∗task)
- int starpu_get_prefetch_flag (void)
- int starpu_prefetch_task_input_on_node_prio (struct starpu_task ∗task, unsigned node, int prio)
- int starpu_prefetch_task_input_on_node (struct starpu_task ∗task, unsigned node)
- int starpu_idle_prefetch_task_input_on_node_prio (struct starpu_task ∗task, unsigned node, int prio)
- int starpu_idle_prefetch_task_input_on_node (struct starpu_task ∗task, unsigned node)
- int starpu_prefetch_task_input_for_prio (struct starpu_task ∗task, unsigned worker, int prio)
- int starpu_prefetch_task_input_for (struct starpu_task ∗task, unsigned worker)
- int starpu_idle_prefetch_task_input_for_prio (struct starpu_task ∗task, unsigned worker, int prio)
- int starpu_idle_prefetch_task_input_for (struct starpu_task ∗task, unsigned worker)
- uint32_t starpu_task_footprint (struct starpu_perfmodel ∗model, struct starpu_task ∗task, struct starpu_perfmodel_arch ∗arch, unsigned nimpl)
- uint32_t starpu_task_data_footprint (struct starpu_task ∗task)
- double starpu_task_expected_length (struct starpu_task ∗task, struct starpu_perfmodel_arch ∗arch, unsigned nimpl)
- double starpu_task_worker_expected_length (struct starpu_task ∗task, unsigned workerid, unsigned sched_ctx_id, unsigned nimpl)
- double starpu_task_expected_length_average (struct starpu_task ∗task, unsigned sched_ctx_id)
- double starpu_worker_get_relative_speedup (struct starpu_perfmodel_arch ∗perf_arch)
- double starpu_task_expected_data_transfer_time (unsigned memory_node, struct starpu_task ∗task)
- double starpu_task_expected_data_transfer_time_for (struct starpu_task ∗task, unsigned worker)
- double starpu_data_expected_transfer_time (starpu_data_handle_t handle, unsigned memory_node, enum starpu_data_access_mode mode)
- double starpu_task_expected_energy (struct starpu_task ∗task, struct starpu_perfmodel_arch ∗arch, unsigned nimpl)
- double starpu_task_worker_expected_energy (struct starpu_task ∗task, unsigned workerid, unsigned sched_ctx_id, unsigned nimpl)
- double starpu_task_expected_energy_average (struct starpu_task ∗task, unsigned sched_ctx_id)
- double starpu_task_expected_conversion_time (struct starpu_task ∗task, struct starpu_perfmodel_arch ∗arch, unsigned nimpl)
- void starpu_task_notify_ready_soon_register (starpu_notify_ready_soon_func f, void ∗data)
- void starpu_sched_ctx_worker_shares_tasks_lists (int workerid, int sched_ctx_id)
- void starpu_sched_task_break (struct starpu_task ∗task)

Worker operations

- int starpu_wake_worker_relax (int workerid)
- int starpu_wake_worker_no_relax (int workerid)
- int starpu_wake_worker_locked (int workerid)
- int starpu_wake_worker_relax_light (int workerid)

59.40 starpu_simgrid_wrap.h File Reference

`#include <starpu_config.h>`

Macros

- #define **main**

59.41 starpu_sink.h File Reference

Functions

- void **starpu_sink_common_worker** (int argc, char ∗∗argv)

59.42 starpu_stdlib.h File Reference

`#include <starpu.h>`

Macros

- #define STARPU_MALLOC_PINNED
- #define STARPU_MALLOC_COUNT
- #define STARPU_MALLOC_NORECLAIM
- #define STARPU_MEMORY_WAIT
- #define STARPU_MEMORY_OVERFLOW
- #define STARPU_MALLOC_SIMULATION_FOLDED
- #define STARPU_MALLOC_SIMULATION_UNIQUE
- #define starpu_data_malloc_pinned_if_possible
- #define starpu_data_free_pinned_if_possible

Typedefs

- typedef int(∗ **starpu_malloc_hook**) (unsigned dst_node, void ∗∗A, size_t dim, int flags)
- typedef int(∗ **starpu_free_hook**) (unsigned dst_node, void ∗A, size_t dim, int flags)

Functions

- void starpu_malloc_set_align (size_t align)
- int starpu_malloc (void ∗∗A, size_t dim)
- int starpu_free (void ∗A)
- int starpu_malloc_flags (void ∗∗A, size_t dim, int flags)
- int starpu_free_flags (void ∗A, size_t dim, int flags)
- int starpu_free_noflag (void ∗A, size_t dim)
- void starpu_malloc_set_hooks (starpu_malloc_hook malloc_hook, starpu_free_hook free_hook)
- int starpu_memory_pin (void ∗addr, size_t size)
- int starpu_memory_unpin (void ∗addr, size_t size)
- starpu_ssize_t starpu_memory_get_total (unsigned node)
- starpu_ssize_t starpu_memory_get_available (unsigned node)
- size_t starpu_memory_get_used (unsigned node)
- starpu_ssize_t starpu_memory_get_total_all_nodes (void)
- starpu_ssize_t starpu_memory_get_available_all_nodes (void)
- size_t starpu_memory_get_used_all_nodes (void)
- int starpu_memory_allocate (unsigned node, size_t size, int flags)

- void starpu_memory_deallocate (unsigned node, size_t size)
- void starpu_memory_wait_available (unsigned node, size_t size)
- void starpu_sleep (float nb_sec)
- void starpu_usleep (float nb_micro_sec)
- void starpu_energy_use (float joules)
- double starpu_energy_used (void)

59.43 starpu_task.h File Reference

```
#include <starpu.h>
#include <errno.h>
#include <assert.h>
#include <cuda.h>
```

Data Structures

- struct starpu_codelet
- struct starpu_data_descr
- struct starpu_task

Macros

- #define STARPU_NOWHERE
- #define STARPU_WORKER_TO_MASK(worker_archtype)
- #define STARPU_CPU
- #define STARPU_CUDA
- #define STARPU_HIP
- #define STARPU_OPENCL
- #define STARPU_MAX_FPGA
- #define STARPU_MPI_MS
- #define STARPU_TCPIP_MS
- #define STARPU_CODELET_SIMGRID_EXECUTE
- #define STARPU_CODELET_SIMGRID_EXECUTE_AND_INJECT
- #define STARPU_CODELET_NOPLANS
- #define STARPU_CUDA_ASYNC
- #define STARPU_HIP_ASYNC
- #define STARPU_OPENCL_ASYNC
- #define STARPU_MAIN_RAM
- #define **STARPU_TASK_INIT**
- #define STARPU_TASK_INVALID
- #define STARPU_MULTIPLE_CPU_IMPLEMENTATIONS
- #define STARPU_MULTIPLE_CUDA_IMPLEMENTATIONS
- #define STARPU_MULTIPLE_HIP_IMPLEMENTATIONS
- #define STARPU_MULTIPLE_OPENCL_IMPLEMENTATIONS
- #define STARPU_VARIABLE_NBUFFERS
- #define STARPU_SPECIFIC_NODE_LOCAL
- #define STARPU_SPECIFIC_NODE_CPU
- #define STARPU_SPECIFIC_NODE_SLOW
- #define STARPU_SPECIFIC_NODE_FAST
- #define STARPU_SPECIFIC_NODE_LOCAL_OR_CPU
- #define STARPU_SPECIFIC_NODE_NONE
- #define STARPU_TASK_TYPE_NORMAL
- #define STARPU_TASK_TYPE_INTERNAL
- #define STARPU_TASK_TYPE_DATA_ACQUIRE

- #define STARPU_TASK_INITIALIZER
- #define STARPU_TASK_GET_NBUFFERS(task)
- #define STARPU_TASK_GET_HANDLE(task, i)
- #define STARPU_TASK_GET_HANDLES(task)
- #define STARPU_TASK_SET_HANDLE(task, handle, i)
- #define STARPU_CODELET_GET_MODE(codelet, i)
- #define STARPU_CODELET_SET_MODE(codelet, mode, i)
- #define STARPU_TASK_GET_MODE(task, i)
- #define STARPU_TASK_SET_MODE(task, mode, i)
- #define STARPU_CODELET_GET_NODE(codelet, i)
- #define STARPU_CODELET_SET_NODE(codelet, __node, i)

Typedefs

- typedef void(∗ starpu_cpu_func_t) (void ∗∗, void ∗)
- typedef void(∗ starpu_cuda_func_t) (void ∗∗, void ∗)
- typedef void(∗ starpu_hip_func_t) (void ∗∗, void ∗)
- typedef void(∗ starpu_opencl_func_t) (void ∗∗, void ∗)
- typedef void(∗ starpu_max_fpga_func_t) (void ∗∗, void ∗)
- typedef int(∗ starpu_bubble_func_t) (struct starpu_task ∗t, void ∗arg)
- typedef void(∗ starpu_bubble_gen_dag_func_t) (struct starpu_task ∗t, void ∗arg)
- typedef struct _starpu_trs_epoch ∗ **starpu_trs_epoch_t**

Enumerations

- enum starpu_codelet_type { STARPU_SEQ , STARPU_SPMD , STARPU_FORKJOIN }
- enum starpu_task_status {
  STARPU_TASK_INIT , STARPU_TASK_INIT , STARPU_TASK_BLOCKED , STARPU_TASK_READY ,
  STARPU_TASK_RUNNING , STARPU_TASK_FINISHED , STARPU_TASK_BLOCKED_ON_TAG ,
  STARPU_TASK_BLOCKED_ON_TASK ,
  STARPU_TASK_BLOCKED_ON_DATA , STARPU_TASK_STOPPED }

Functions

- void starpu_task_init (struct starpu_task ∗task)
- void starpu_task_clean (struct starpu_task ∗task)
- struct starpu_task ∗ starpu_task_create (void) STARPU_ATTRIBUTE_MALLOC
- struct starpu_task ∗ starpu_task_create_sync (starpu_data_handle_t handle, enum starpu_data_access_mode mode) STARPU_ATTRIBUTE_MALLOC
- void starpu_task_destroy (struct starpu_task ∗task)
- void starpu_task_set_destroy (struct starpu_task ∗task)
- int starpu_task_submit (struct starpu_task ∗task)
- int starpu_task_submit_nodeps (struct starpu_task ∗task)
- int starpu_task_submit_to_ctx (struct starpu_task ∗task, unsigned sched_ctx_id)
- int starpu_task_finished (struct starpu_task ∗task)
- int starpu_task_wait (struct starpu_task ∗task)
- int starpu_task_wait_array (struct starpu_task ∗∗tasks, unsigned nb_tasks)
- int starpu_task_wait_for_all (void)
- int starpu_task_wait_for_n_submitted (unsigned n)
- int starpu_task_wait_for_all_in_ctx (unsigned sched_ctx_id)
- int starpu_task_wait_for_n_submitted_in_ctx (unsigned sched_ctx_id, unsigned n)
- int starpu_task_wait_for_no_ready (void)
- int starpu_task_nready (void)
- int starpu_task_nsubmitted (void)
- void starpu_iteration_push (unsigned long iteration)
- void starpu_iteration_pop (void)

- void starpu_do_schedule (void)
- void starpu_codelet_init (struct starpu_codelet ∗cl)
- void starpu_codelet_display_stats (struct starpu_codelet ∗cl)
- struct starpu_task ∗ starpu_task_get_current (void)
- int starpu_task_get_current_data_node (unsigned i)
- const char ∗ starpu_task_get_model_name (struct starpu_task ∗task)
- const char ∗ starpu_task_get_name (struct starpu_task ∗task)
- struct starpu_task ∗ starpu_task_dup (struct starpu_task ∗task)
- void starpu_task_set_implementation (struct starpu_task ∗task, unsigned impl)
- unsigned starpu_task_get_implementation (struct starpu_task ∗task)
- void starpu_create_sync_task (starpu_tag_t sync_tag, unsigned ndeps, starpu_tag_t ∗deps, void(∗callback)(void ∗), void ∗callback_arg)
- void starpu_create_callback_task (void(∗callback)(void ∗), void ∗callback_arg)
- void starpu_task_ft_prologue (void ∗check_ft)
- struct starpu_task ∗ starpu_task_ft_create_retry (const struct starpu_task ∗meta_task, const struct starpu_task ∗template_task, void(∗check_ft)(void ∗))
- void starpu_task_ft_failed (struct starpu_task ∗task)
- void starpu_task_ft_success (struct starpu_task ∗meta_task)
- void starpu_task_watchdog_set_hook (void(∗hook)(void ∗), void ∗hook_arg)
- char ∗ starpu_task_status_get_as_string (enum starpu_task_status status)
- void starpu_set_limit_min_submitted_tasks (int limit_min)
- void starpu_set_limit_max_submitted_tasks (int limit_min)
- struct starpu_transaction ∗ starpu_transaction_open (int(∗do_start_func)(void ∗buffer, void ∗arg), void ∗do_start_arg)
- void starpu_transaction_next_epoch (struct starpu_transaction ∗p_trs, void ∗do_start_arg)
- void starpu_transaction_close (struct starpu_transaction ∗p_trs)

Variables

- struct starpu_codelet starpu_codelet_nop

59.43.1 Macro Definition Documentation

59.43.1.1 STARPU_TASK_INVALID

```
#define STARPU_TASK_INVALID
```

old name for STARPU_TASK_INIT

59.44 starpu_task_bundle.h File Reference

Typedefs

- typedef struct _starpu_task_bundle ∗ starpu_task_bundle_t

Functions

- void starpu_task_bundle_create (starpu_task_bundle_t ∗bundle)
- int starpu_task_bundle_insert (starpu_task_bundle_t bundle, struct starpu_task ∗task)
- int starpu_task_bundle_remove (starpu_task_bundle_t bundle, struct starpu_task ∗task)
- void starpu_task_bundle_close (starpu_task_bundle_t bundle)
- double starpu_task_bundle_expected_length (starpu_task_bundle_t bundle, struct starpu_perfmodel_arch ∗arch, unsigned nimpl)
- double starpu_task_bundle_expected_data_transfer_time (starpu_task_bundle_t bundle, unsigned memory_node)
- double starpu_task_bundle_expected_energy (starpu_task_bundle_t bundle, struct starpu_perfmodel_arch ∗arch, unsigned nimpl)

59.45 starpu_task_dep.h File Reference

`#include <starpu.h>`

Typedefs

- typedef uint64_t starpu_tag_t

Functions

- void starpu_task_declare_deps_array (struct starpu_task ∗task, unsigned ndeps, struct starpu_task ∗task_array[ ])
- void starpu_task_declare_deps (struct starpu_task ∗task, unsigned ndeps,...)
- void starpu_task_declare_end_deps_array (struct starpu_task ∗task, unsigned ndeps, struct starpu_task ∗task_array[ ])
- void starpu_task_declare_end_deps (struct starpu_task ∗task, unsigned ndeps,...)
- int starpu_task_get_task_succs (struct starpu_task ∗task, unsigned ndeps, struct starpu_task ∗task_array[ ])
- int starpu_task_get_task_scheduled_succs (struct starpu_task ∗task, unsigned ndeps, struct starpu_task ∗task_array[ ])
- void starpu_task_end_dep_add (struct starpu_task ∗t, int nb_deps)
- void starpu_task_end_dep_release (struct starpu_task ∗t)
- void starpu_tag_declare_deps (starpu_tag_t id, unsigned ndeps,...)
- void starpu_tag_declare_deps_array (starpu_tag_t id, unsigned ndeps, starpu_tag_t ∗array)
- int starpu_tag_wait (starpu_tag_t id)
- int starpu_tag_wait_array (unsigned ntags, starpu_tag_t ∗id)
- void starpu_tag_restart (starpu_tag_t id)
- void starpu_tag_remove (starpu_tag_t id)
- void starpu_tag_notify_from_apps (starpu_tag_t id)
- void starpu_tag_notify_restart_from_apps (starpu_tag_t id)
- struct starpu_task ∗ starpu_tag_get_task (starpu_tag_t id)

59.46 starpu_task_list.h File Reference

`#include <starpu_task.h>`
`#include <starpu_util.h>`

Data Structures

- struct starpu_task_list

Functions

- void starpu_task_list_init (struct starpu_task_list ∗list)
- void starpu_task_list_push_front (struct starpu_task_list ∗list, struct starpu_task ∗task)
- void starpu_task_list_push_back (struct starpu_task_list ∗list, struct starpu_task ∗task)
- struct starpu_task ∗ starpu_task_list_front (const struct starpu_task_list ∗list)
- struct starpu_task ∗ starpu_task_list_back (const struct starpu_task_list ∗list)
- int starpu_task_list_empty (const struct starpu_task_list ∗list)
- void starpu_task_list_erase (struct starpu_task_list ∗list, struct starpu_task ∗task)
- struct starpu_task ∗ starpu_task_list_pop_front (struct starpu_task_list ∗list)
- struct starpu_task ∗ starpu_task_list_pop_back (struct starpu_task_list ∗list)
- struct starpu_task ∗ starpu_task_list_begin (const struct starpu_task_list ∗list)
- struct starpu_task ∗ starpu_task_list_end (const struct starpu_task_list ∗list STARPU_ATTRIBUTE_UNUSED)
- struct starpu_task ∗ starpu_task_list_next (const struct starpu_task ∗task)
- int starpu_task_list_ismember (const struct starpu_task_list ∗list, const struct starpu_task ∗look)
- void starpu_task_list_move (struct starpu_task_list ∗ldst, struct starpu_task_list ∗lsrc)

59.47 starpu_task_util.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <starpu.h>
```

Data Structures

- struct starpu_codelet_pack_arg_data

Macros

- #define **STARPU_MODE_SHIFT**
- #define STARPU_VALUE
- #define STARPU_CALLBACK
- #define STARPU_CALLBACK_WITH_ARG
- #define STARPU_CALLBACK_ARG
- #define STARPU_PRIORITY
- #define STARPU_EXECUTE_ON_NODE
- #define STARPU_EXECUTE_ON_DATA
- #define STARPU_DATA_ARRAY
- #define STARPU_DATA_MODE_ARRAY
- #define STARPU_TAG
- #define STARPU_HYPERVISOR_TAG
- #define STARPU_FLOPS
- #define STARPU_SCHED_CTX
- #define STARPU_PROLOGUE_CALLBACK
- #define STARPU_PROLOGUE_CALLBACK_ARG
- #define STARPU_PROLOGUE_CALLBACK_POP
- #define STARPU_PROLOGUE_CALLBACK_POP_ARG
- #define STARPU_EXECUTE_ON_WORKER
- #define STARPU_EXECUTE_WHERE
- #define STARPU_TAG_ONLY
- #define STARPU_POSSIBLY_PARALLEL
- #define STARPU_WORKER_ORDER
- #define STARPU_NODE_SELECTION_POLICY
- #define STARPU_NAME
- #define STARPU_CL_ARGS
- #define STARPU_CL_ARGS_NFREE
- #define STARPU_TASK_DEPS_ARRAY
- #define STARPU_TASK_COLOR
- #define STARPU_HANDLES_SEQUENTIAL_CONSISTENCY
- #define STARPU_TASK_SYNCHRONOUS
- #define STARPU_TASK_END_DEPS_ARRAY
- #define STARPU_TASK_END_DEP
- #define STARPU_TASK_WORKERIDS
- #define STARPU_SEQUENTIAL_CONSISTENCY
- #define STARPU_TASK_PROFILING_INFO
- #define STARPU_TASK_NO_SUBMITORDER
- #define STARPU_CALLBACK_ARG_NFREE
- #define STARPU_CALLBACK_WITH_ARG_NFREE
- #define STARPU_PROLOGUE_CALLBACK_ARG_NFREE
- #define STARPU_PROLOGUE_CALLBACK_POP_ARG_NFREE

- #define STARPU_TASK_SCHED_DATA
- #define STARPU_TRANSACTION
- #define STARPU_TASK_FILE
- #define STARPU_TASK_LINE
- #define STARPU_EPILOGUE_CALLBACK
- #define STARPU_EPILOGUE_CALLBACK_ARG
- #define STARPU_BUBBLE_FUNC
- #define STARPU_BUBBLE_FUNC_ARG
- #define STARPU_BUBBLE_GEN_DAG_FUNC
- #define STARPU_BUBBLE_GEN_DAG_FUNC_ARG
- #define STARPU_BUBBLE_PARENT
- #define STARPU_SHIFTED_MODE_MAX

Functions

- int starpu_task_set (struct starpu_task ∗task, struct starpu_codelet ∗cl,...)
- struct starpu_task ∗ starpu_task_build (struct starpu_codelet ∗cl,...)
- int starpu_task_insert (struct starpu_codelet ∗cl,...)
- int starpu_insert_task (struct starpu_codelet ∗cl,...)
- void starpu_task_insert_data_make_room (struct starpu_codelet ∗cl, struct starpu_task ∗task, int ∗allocated_buffers, int current_buffer, int room)
- void starpu_task_insert_data_process_arg (struct starpu_codelet ∗cl, struct starpu_task ∗task, int ∗allocated_buffers, int ∗current_buffer, int arg_type, starpu_data_handle_t handle)
- void starpu_task_insert_data_process_array_arg (struct starpu_codelet ∗cl, struct starpu_task ∗task, int ∗allocated_buffers, int ∗current_buffer, int nb_handles, starpu_data_handle_t ∗handles)
- void starpu_task_insert_data_process_mode_array_arg (struct starpu_codelet ∗cl, struct starpu_task ∗task, int ∗allocated_buffers, int ∗current_buffer, int nb_descrs, struct starpu_data_descr ∗descrs)
- void starpu_codelet_pack_args (void ∗∗arg_buffer, size_t ∗arg_buffer_size,...)
- void starpu_codelet_pack_arg_init (struct starpu_codelet_pack_arg_data ∗state)
- void starpu_codelet_pack_arg (struct starpu_codelet_pack_arg_data ∗state, const void ∗ptr, size_t ptr_size)
- void starpu_codelet_pack_arg_fini (struct starpu_codelet_pack_arg_data ∗state, void ∗∗cl_arg, size_t ∗cl_arg_size)
- void starpu_codelet_unpack_args (void ∗cl_arg,...)
- void starpu_codelet_unpack_arg_init (struct starpu_codelet_pack_arg_data ∗state, void ∗cl_arg, size_t cl_arg_size)
- void starpu_codelet_unpack_arg (struct starpu_codelet_pack_arg_data ∗state, void ∗ptr, size_t size)
- void starpu_codelet_dup_arg (struct starpu_codelet_pack_arg_data ∗state, void ∗∗ptr, size_t ∗size)
- void starpu_codelet_pick_arg (struct starpu_codelet_pack_arg_data ∗state, void ∗∗ptr, size_t ∗size)
- void starpu_codelet_unpack_arg_fini (struct starpu_codelet_pack_arg_data ∗state)
- void starpu_codelet_unpack_discard_arg (struct starpu_codelet_pack_arg_data ∗state)
- void starpu_codelet_unpack_args_and_copyleft (void ∗cl_arg, void ∗buffer, size_t buffer_size,...)

59.48 starpu_thread.h File Reference

```
#include <starpu_config.h>
#include <starpu_util.h>
#include <pthread.h>
#include <xbt/synchro_core.h>
#include <stdint.h>
```

Data Structures

- struct starpu_pthread_attr_t
- struct starpu_pthread_barrier_t
- struct starpu_pthread_spinlock_t
- struct starpu_pthread_wait_t
- struct starpu_pthread_queue_t

Macros

- #define **starpu_pthread_setname**(name)
- #define STARPU_PTHREAD_MUTEX_INITIALIZER
- #define STARPU_PTHREAD_COND_INITIALIZER
- #define **STARPU_PTHREAD_RWLOCK_INITIALIZER**
- #define **STARPU_PTHREAD_BARRIER_SERIAL_THREAD**

Typedefs

- typedef msg_process_t **starpu_pthread_t**
- typedef msg_host_t **starpu_sg_host_t**
- typedef xbt_mutex_t **starpu_pthread_mutex_t**
- typedef int **starpu_pthread_mutexattr_t**
- typedef int **starpu_pthread_key_t**
- typedef xbt_cond_t **starpu_pthread_cond_t**
- typedef int **starpu_pthread_condattr_t**
- typedef xbt_mutex_t **starpu_pthread_rwlock_t**
- typedef int **starpu_pthread_rwlockattr_t**
- typedef int **starpu_pthread_barrierattr_t**
- typedef msg_sem_t **starpu_sem_t**

Functions

- int **starpu_pthread_equal** (starpu_pthread_t t1, starpu_pthread_t t2)
- starpu_pthread_t **starpu_pthread_self** (void)
- int **starpu_pthread_create_on** (const char ∗name, starpu_pthread_t ∗thread, const starpu_pthread_attr_t ∗attr, void ∗(∗start_routine)(void ∗), void ∗arg, starpu_sg_host_t host)
- int starpu_pthread_create (starpu_pthread_t ∗thread, const starpu_pthread_attr_t ∗attr, void ∗(∗start_routine)(void ∗), void ∗arg)
- starpu_pthread_t **_starpu_simgrid_actor_create** (const char ∗name, xbt_main_func_t code, starpu_sg_host_t host, int argc, char ∗argv[ ])
- int starpu_pthread_join (starpu_pthread_t thread, void ∗∗retval)
- int **starpu_pthread_detach** (starpu_pthread_t thread)
- int starpu_pthread_exit (void ∗retval) STARPU_ATTRIBUTE_NORETURN
- int starpu_pthread_attr_init (starpu_pthread_attr_t ∗attr)
- int starpu_pthread_attr_destroy (starpu_pthread_attr_t ∗attr)
- int starpu_pthread_attr_setdetachstate (starpu_pthread_attr_t ∗attr, int detachstate)
- int **starpu_pthread_attr_setstacksize** (starpu_pthread_attr_t ∗attr, size_t stacksize)
- int starpu_pthread_mutex_init (starpu_pthread_mutex_t ∗mutex, const starpu_pthread_mutexattr_t ∗mutexattr)
- int starpu_pthread_mutex_destroy (starpu_pthread_mutex_t ∗mutex)
- int starpu_pthread_mutex_lock (starpu_pthread_mutex_t ∗mutex)
- int starpu_pthread_mutex_unlock (starpu_pthread_mutex_t ∗mutex)
- int starpu_pthread_mutex_trylock (starpu_pthread_mutex_t ∗mutex)
- int starpu_pthread_mutexattr_gettype (const starpu_pthread_mutexattr_t ∗attr, int ∗type)
- int starpu_pthread_mutexattr_settype (starpu_pthread_mutexattr_t ∗attr, int type)
- int starpu_pthread_mutexattr_destroy (starpu_pthread_mutexattr_t ∗attr)

- int starpu_pthread_mutexattr_init (starpu_pthread_mutexattr_t ∗attr)
- int **starpu_pthread_mutex_lock_sched** (starpu_pthread_mutex_t ∗mutex)
- int **starpu_pthread_mutex_unlock_sched** (starpu_pthread_mutex_t ∗mutex)
- int **starpu_pthread_mutex_trylock_sched** (starpu_pthread_mutex_t ∗mutex)
- void **starpu_pthread_mutex_check_sched** (starpu_pthread_mutex_t ∗mutex, char ∗file, int line)
- int starpu_pthread_key_create (starpu_pthread_key_t ∗key, void(∗destr_function)(void ∗))
- int starpu_pthread_key_delete (starpu_pthread_key_t key)
- int starpu_pthread_setspecific (starpu_pthread_key_t key, const void ∗pointer)
- void ∗ starpu_pthread_getspecific (starpu_pthread_key_t key)
- int starpu_pthread_cond_init (starpu_pthread_cond_t ∗cond, starpu_pthread_condattr_t ∗cond_attr)
- int starpu_pthread_cond_signal (starpu_pthread_cond_t ∗cond)
- int starpu_pthread_cond_broadcast (starpu_pthread_cond_t ∗cond)
- int starpu_pthread_cond_wait (starpu_pthread_cond_t ∗cond, starpu_pthread_mutex_t ∗mutex)
- int starpu_pthread_cond_timedwait (starpu_pthread_cond_t ∗cond, starpu_pthread_mutex_t ∗mutex, const struct timespec ∗abstime)
- int starpu_pthread_cond_destroy (starpu_pthread_cond_t ∗cond)
- int starpu_pthread_rwlock_init (starpu_pthread_rwlock_t ∗rwlock, const starpu_pthread_rwlockattr_t ∗attr)
- int starpu_pthread_rwlock_destroy (starpu_pthread_rwlock_t ∗rwlock)
- int starpu_pthread_rwlock_rdlock (starpu_pthread_rwlock_t ∗rwlock)
- int starpu_pthread_rwlock_tryrdlock (starpu_pthread_rwlock_t ∗rwlock)
- int starpu_pthread_rwlock_wrlock (starpu_pthread_rwlock_t ∗rwlock)
- int starpu_pthread_rwlock_trywrlock (starpu_pthread_rwlock_t ∗rwlock)
- int starpu_pthread_rwlock_unlock (starpu_pthread_rwlock_t ∗rwlock)
- int starpu_pthread_barrier_init (starpu_pthread_barrier_t ∗barrier, const starpu_pthread_barrierattr_t ∗attr, unsigned count)
- int starpu_pthread_barrier_destroy (starpu_pthread_barrier_t ∗barrier)
- int starpu_pthread_barrier_wait (starpu_pthread_barrier_t ∗barrier)
- int starpu_pthread_spin_init (starpu_pthread_spinlock_t ∗lock, int pshared)
- int starpu_pthread_spin_destroy (starpu_pthread_spinlock_t ∗lock)
- int starpu_pthread_spin_lock (starpu_pthread_spinlock_t ∗lock)
- int starpu_pthread_spin_trylock (starpu_pthread_spinlock_t ∗lock)
- int starpu_pthread_spin_unlock (starpu_pthread_spinlock_t ∗lock)
- int **starpu_pthread_queue_init** (starpu_pthread_queue_t ∗q)
- int **starpu_pthread_queue_signal** (starpu_pthread_queue_t ∗q)
- int **starpu_pthread_queue_broadcast** (starpu_pthread_queue_t ∗q)
- int **starpu_pthread_queue_destroy** (starpu_pthread_queue_t ∗q)
- int **starpu_pthread_wait_init** (starpu_pthread_wait_t ∗w)
- int **starpu_pthread_queue_register** (starpu_pthread_wait_t ∗w, starpu_pthread_queue_t ∗q)
- int **starpu_pthread_queue_unregister** (starpu_pthread_wait_t ∗w, starpu_pthread_queue_t ∗q)
- int **starpu_pthread_wait_reset** (starpu_pthread_wait_t ∗w)
- int **starpu_pthread_wait_wait** (starpu_pthread_wait_t ∗w)
- int **starpu_pthread_wait_timedwait** (starpu_pthread_wait_t ∗w, const struct timespec ∗abstime)
- int **starpu_pthread_wait_destroy** (starpu_pthread_wait_t ∗w)
- int **starpu_sem_destroy** (starpu_sem_t ∗sem)
- int **starpu_sem_getvalue** (starpu_sem_t ∗sem, int ∗retval)
- int **starpu_sem_init** (starpu_sem_t ∗sem, int pshared, unsigned value)
- int **starpu_sem_post** (starpu_sem_t ∗sem)
- int **starpu_sem_trywait** (starpu_sem_t ∗sem)
- int **starpu_sem_wait** (starpu_sem_t ∗sem)

59.48.1 Data Structure Documentation

59.48.1.1 struct starpu_pthread_attr_t

Data Fields

size_t	stacksize	

59.48.1.2 struct starpu_pthread_barrier_t

Data Fields

starpu_pthread_mutex_t	mutex	
starpu_pthread_cond_t	cond	
starpu_pthread_cond_t	cond_destroy	
unsigned	count	
unsigned	done	
unsigned	busy	

59.48.1.3 struct starpu_pthread_spinlock_t

Data Fields

int	taken	

59.48.1.4 struct starpu_pthread_wait_t

Data Fields

starpu_pthread_mutex_t	mutex	
starpu_pthread_cond_t	cond	
unsigned	block	

59.48.1.5 struct starpu_pthread_queue_t

Data Fields

starpu_pthread_mutex_t	mutex	
starpu_pthread_wait_t **	queue	
unsigned	allocqueue	
unsigned	nqueue	

59.49 starpu_thread_util.h File Reference

```
#include <starpu_util.h>
#include <starpu_thread.h>
#include <errno.h>
```

Macros

- #define STARPU_PTHREAD_CREATE_ON(name, thread, attr, routine, arg, where)
- #define STARPU_PTHREAD_CREATE(thread, attr, routine, arg)
- #define **STARPU_PTHREAD_JOIN**(thread, retval)
- #define **_STARPU_PTHREAD_MUTEX_INIT**(mutex, attr)
- #define STARPU_PTHREAD_MUTEX_INIT(mutex, attr)

- #define STARPU_PTHREAD_MUTEX_INIT0(mutex, attr)
- #define STARPU_PTHREAD_MUTEX_DESTROY(mutex)
- #define **_STARPU_CHECK_NOT_SCHED_MUTEX**(mutex, file, line)
- #define STARPU_PTHREAD_MUTEX_LOCK(mutex)
- #define **STARPU_PTHREAD_MUTEX_LOCK_SCHED**(mutex)
- #define **STARPU_PTHREAD_MUTEX_TRYLOCK**(mutex)
- #define **STARPU_PTHREAD_MUTEX_TRYLOCK_SCHED**(mutex)
- #define STARPU_PTHREAD_MUTEX_UNLOCK(mutex)
- #define **STARPU_PTHREAD_MUTEX_UNLOCK_SCHED**(mutex)
- #define STARPU_PTHREAD_KEY_CREATE(key, destr)
- #define STARPU_PTHREAD_KEY_DELETE(key)
- #define STARPU_PTHREAD_SETSPECIFIC(key, ptr)
- #define STARPU_PTHREAD_GETSPECIFIC(key)
- #define **_STARPU_PTHREAD_RWLOCK_INIT**(rwlock, attr)
- #define STARPU_PTHREAD_RWLOCK_INIT(rwlock, attr)
- #define STARPU_PTHREAD_RWLOCK_INIT0(rwlock, attr)
- #define STARPU_PTHREAD_RWLOCK_RDLOCK(rwlock)
- #define **STARPU_PTHREAD_RWLOCK_TRYRDLOCK**(rwlock)
- #define STARPU_PTHREAD_RWLOCK_WRLOCK(rwlock)
- #define **STARPU_PTHREAD_RWLOCK_TRYWRLOCK**(rwlock)
- #define STARPU_PTHREAD_RWLOCK_UNLOCK(rwlock)
- #define STARPU_PTHREAD_RWLOCK_DESTROY(rwlock)
- #define **_STARPU_PTHREAD_COND_INIT**(cond, attr)
- #define STARPU_PTHREAD_COND_INIT(cond, attr)
- #define STARPU_PTHREAD_COND_INIT0(cond, attr)
- #define STARPU_PTHREAD_COND_DESTROY(cond)
- #define STARPU_PTHREAD_COND_SIGNAL(cond)
- #define STARPU_PTHREAD_COND_BROADCAST(cond)
- #define STARPU_PTHREAD_COND_WAIT(cond, mutex)
- #define **STARPU_PTHREAD_COND_TIMEDWAIT**(cond, mutex, abstime)
- #define STARPU_PTHREAD_BARRIER_INIT(barrier, attr, count)
- #define STARPU_PTHREAD_BARRIER_DESTROY(barrier)
- #define STARPU_PTHREAD_BARRIER_WAIT(barrier)

Functions

- static STARPU_INLINE int **_starpu_pthread_mutex_trylock** (starpu_pthread_mutex_t ∗mutex, char ∗file, int line)
- static STARPU_INLINE int **_starpu_pthread_mutex_trylock_sched** (starpu_pthread_mutex_t ∗mutex, char ∗file, int line)
- static STARPU_INLINE int **_starpu_pthread_rwlock_tryrdlock** (starpu_pthread_rwlock_t ∗rwlock, char ∗file, int line)
- static STARPU_INLINE int **_starpu_pthread_rwlock_trywrlock** (starpu_pthread_rwlock_t ∗rwlock, char ∗file, int line)
- static STARPU_INLINE int **_starpu_pthread_cond_timedwait** (starpu_pthread_cond_t ∗cond, starpu_pthread_mutex_t ∗mutex, const struct timespec ∗abstime, char ∗file, int line)

59.50 starpu_tree.h File Reference

Data Structures

- struct starpu_tree

Functions

- void **starpu_tree_reset_visited** (struct starpu_tree ∗tree, char ∗visited)
- void **starpu_tree_prepare_children** (unsigned arity, struct starpu_tree ∗father)
- void **starpu_tree_insert** (struct starpu_tree ∗tree, int id, int level, int is_pu, int arity, struct starpu_tree ∗father)
- struct starpu_tree ∗ **starpu_tree_get** (struct starpu_tree ∗tree, int id)
- struct starpu_tree ∗ **starpu_tree_get_neighbour** (struct starpu_tree ∗tree, struct starpu_tree ∗node, char ∗visited, char ∗present)
- void **starpu_tree_free** (struct starpu_tree ∗tree)

59.51 starpu_util.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <assert.h>
#include <starpu_config.h>
#include <sys/time.h>
```

Macros

- #define STARPU_GNUC_PREREQ(maj, min)
- #define STARPU_UNLIKELY(expr)
- #define STARPU_LIKELY(expr)
- #define STARPU_ATTRIBUTE_UNUSED
- #define STARPU_ATTRIBUTE_NORETURN
- #define STARPU_ATTRIBUTE_VISIBILITY_DEFAULT
- #define STARPU_VISIBILITY_PUSH_HIDDEN
- #define STARPU_VISIBILITY_POP
- #define STARPU_ATTRIBUTE_MALLOC
- #define STARPU_ATTRIBUTE_WARN_UNUSED_RESULT
- #define STARPU_ATTRIBUTE_PURE
- #define STARPU_ATTRIBUTE_ALIGNED(size)
- #define **STARPU_ATTRIBUTE_FORMAT**(type, string, first)
- #define **STARPU_INLINE**
- #define **STARPU_ATTRIBUTE_CALLOC_SIZE**(num, size)
- #define **STARPU_ATTRIBUTE_ALLOC_SIZE**(size)
- #define **endif**
- #define **endif**
- #define **STARPU_BACKTRACE_LENGTH**
- #define **STARPU_DUMP_BACKTRACE**()
- #define **STARPU_SIMGRID_ASSERT**(x)
- #define STARPU_ASSERT(x)
- #define STARPU_ASSERT_ACCESSIBLE(ptr)
- #define STARPU_STATIC_ASSERT(x)
- #define STARPU_ASSERT_MSG(x, msg, ...)
- #define **_starpu_abort**()
- #define STARPU_ABORT()
- #define STARPU_ABORT_MSG(msg, ...)
- #define STARPU_CHECK_RETURN_VALUE(err, message, ...)
- #define STARPU_CHECK_RETURN_VALUE_IS(err, value, message, ...)
- #define **STARPU_ATOMIC_SOMETHING**(name, expr)
- #define **STARPU_ATOMIC_SOMETHINGL**(name, expr)

- #define **STARPU_ATOMIC_SOMETHING64**(name, expr)
- #define **STARPU_BOOL_COMPARE_AND_SWAP_PTR**(ptr, old, value)
- #define **STARPU_VAL_COMPARE_AND_SWAP_PTR**(ptr, old, value)
- #define STARPU_RMB()
- #define STARPU_WMB()
- #define **STARPU_CACHELINE_SIZE**

59.52 starpu_worker.h File Reference

```
#include <stdlib.h>
#include <starpu_config.h>
#include <starpu_thread.h>
#include <starpu_task.h>
#include <hwloc.h>
```

Data Structures

- struct starpu_sched_ctx_iterator
- struct starpu_worker_collection

Macros

- #define STARPU_UNKNOWN_WORKER
- #define starpu_worker_get_id_check()

Enumerations

- enum starpu_node_kind {
  STARPU_UNUSED , STARPU_CPU_RAM , STARPU_CUDA_RAM , STARPU_OPENCL_RAM ,
  STARPU_MAX_FPGA_RAM , STARPU_DISK_RAM , STARPU_MPI_MS_RAM , STARPU_TCPIP_MS_RAM
  ,
  STARPU_HIP_RAM , STARPU_MAX_RAM , STARPU_NRAM }
- enum starpu_worker_archtype {
  STARPU_CPU_WORKER , STARPU_CUDA_WORKER , STARPU_OPENCL_WORKER , STARPU_MAX_FPGA_WORKER
  ,
  STARPU_MPI_MS_WORKER , STARPU_TCPIP_MS_WORKER , STARPU_HIP_WORKER , STARPU_NARCH
  ,
  STARPU_ANY_WORKER }
- enum starpu_worker_collection_type { STARPU_WORKER_TREE , STARPU_WORKER_LIST }

Functions

- void starpu_worker_wait_for_initialisation (void)
- unsigned starpu_worker_archtype_is_valid (enum starpu_worker_archtype type)
- enum starpu_worker_archtype starpu_arch_mask_to_worker_archtype (unsigned mask)
- unsigned starpu_worker_get_count (void)
- unsigned starpu_cpu_worker_get_count (void)
- unsigned starpu_cuda_worker_get_count (void)
- unsigned starpu_hip_worker_get_count (void)
- unsigned starpu_opencl_worker_get_count (void)
- unsigned starpu_mpi_ms_worker_get_count (void)
- unsigned starpu_tcpip_ms_worker_get_count (void)
- int starpu_worker_get_id (void)
- unsigned **_starpu_worker_get_id_check** (const char ∗f, int l)
- int starpu_worker_get_bindid (int workerid)

- void starpu_sched_find_all_worker_combinations (void)
- enum starpu_worker_archtype starpu_worker_get_type (int id)
- int starpu_worker_get_count_by_type (enum starpu_worker_archtype type)
- unsigned starpu_worker_get_ids_by_type (enum starpu_worker_archtype type, int ∗workerids, unsigned maxsize)
- int starpu_worker_get_by_type (enum starpu_worker_archtype type, int num)
- int starpu_worker_get_by_devid (enum starpu_worker_archtype type, int devid)
- unsigned starpu_worker_type_can_execute_task (enum starpu_worker_archtype worker_type, const struct starpu_task ∗task)
- void starpu_worker_get_name (int id, char ∗dst, size_t maxlen)
- void starpu_worker_display_all (FILE ∗output)
- void starpu_worker_display_names (FILE ∗output, enum starpu_worker_archtype type)
- void starpu_worker_display_count (FILE ∗output, enum starpu_worker_archtype type)
- int starpu_worker_get_devid (int id)
- int starpu_worker_get_devnum (int id)
- int starpu_worker_get_subworkerid (int id)
- struct starpu_tree ∗ starpu_workers_get_tree (void)
- unsigned starpu_worker_get_sched_ctx_list (int worker, unsigned ∗∗sched_ctx)
- void starpu_worker_get_current_task_exp_end (unsigned workerid, struct timespec ∗date)
- unsigned starpu_worker_is_blocked_in_parallel (int workerid)
- unsigned starpu_worker_is_slave_somewhere (int workerid)
- const char ∗ starpu_worker_get_type_as_string (enum starpu_worker_archtype type)
- enum starpu_worker_archtype starpu_worker_get_type_from_string (const char ∗type)
- const char ∗ starpu_worker_get_type_as_env_var (enum starpu_worker_archtype type)
- int starpu_bindid_get_workerids (int bindid, int ∗∗workerids)
- int starpu_worker_get_devids (enum starpu_worker_archtype type, int ∗devids, int num)
- int starpu_worker_get_stream_workerids (unsigned devid, int ∗workerids, enum starpu_worker_archtype type)
- hwloc_cpuset_t starpu_worker_get_hwloc_cpuset (int workerid)
- hwloc_obj_t starpu_worker_get_hwloc_obj (int workerid)
- int starpu_memory_node_get_devid (unsigned node)
- unsigned starpu_worker_get_local_memory_node (void)
- unsigned starpu_worker_get_memory_node (unsigned workerid)
- unsigned starpu_memory_nodes_get_count (void)
- unsigned starpu_memory_nodes_get_count_by_kind (enum starpu_node_kind kind)
- unsigned starpu_memory_node_get_ids_by_type (enum starpu_node_kind kind, unsigned ∗memory_nodes_ids, unsigned maxsize)
- int starpu_memory_node_get_name (unsigned node, char ∗name, size_t size)
- unsigned starpu_memory_nodes_get_numa_count (void)
- int starpu_memory_nodes_numa_id_to_devid (int osid)
- int starpu_memory_nodes_numa_devid_to_id (unsigned id)
- enum starpu_node_kind starpu_node_get_kind (unsigned node)
- enum starpu_worker_archtype starpu_memory_node_get_worker_archtype (enum starpu_node_kind node_kind)
- enum starpu_node_kind starpu_worker_get_memory_node_kind (enum starpu_worker_archtype type)
- unsigned starpu_combined_worker_get_count (void)
- unsigned starpu_worker_is_combined_worker (int id)
- int starpu_combined_worker_get_id (void)
- int starpu_combined_worker_get_size (void)
- int starpu_combined_worker_get_rank (void)
- int starpu_combined_worker_assign_workerid (int nworkers, int workerid_array[ ])
- int starpu_combined_worker_get_description (int workerid, int ∗worker_size, int ∗∗combined_workerid)
- int starpu_combined_worker_can_execute_task (unsigned workerid, struct starpu_task ∗task, unsigned nimpl)
- void starpu_parallel_task_barrier_init (struct starpu_task ∗task, int workerid)

- void starpu_parallel_task_barrier_init_n (struct starpu_task ∗task, int worker_size)

Scheduling operations

- int starpu_worker_sched_op_pending (void)
- void starpu_worker_relax_on (void)
- void starpu_worker_relax_off (void)
- int starpu_worker_get_relax_state (void)
- void starpu_worker_lock (int workerid)
- int starpu_worker_trylock (int workerid)
- void starpu_worker_unlock (int workerid)
- void starpu_worker_lock_self (void)
- void starpu_worker_unlock_self (void)
- void starpu_worker_set_going_to_sleep_callback (void(∗callback)(unsigned workerid))
- void starpu_worker_set_waking_up_callback (void(∗callback)(unsigned workerid))

Variables

- struct starpu_worker_collection **starpu_worker_list**
- struct starpu_worker_collection **starpu_worker_tree**

59.53 starpufft.h File Reference

Typedefs

- typedef double _Complex **starpufft_complex**
- typedef struct starpufft_plan ∗ **starpufft_plan**
- typedef float _Complex **starpufftf_complex**
- typedef struct starpufftf_plan ∗ **starpufftf_plan**
- typedef long double _Complex **starpufftl_complex**
- typedef struct starpufftl_plan ∗ **starpufftl_plan**

Functions

- starpufft_plan starpufft_plan_dft_1d (int n, int sign, unsigned flags)
- starpufft_plan starpufft_plan_dft_2d (int n, int m, int sign, unsigned flags)
- starpufft_plan **starpufft_plan_dft_3d** (int n, int m, int p, int sign, unsigned flags)
- starpufft_plan **starpufft_plan_dft_r2c_1d** (int n, unsigned flags)
- starpufft_plan **starpufft_plan_dft_c2r_1d** (int n, unsigned flags)
- void ∗ starpufft_malloc (size_t n)
- void **starpufft_free** (void ∗p, size_t dim)
- int starpufft_execute (starpufft_plan p, void ∗in, void ∗out)
- struct starpu_task ∗ starpufft_start (starpufft_plan p, void ∗in, void ∗out)
- int starpufft_execute_handle (starpufft_plan p, starpu_data_handle_t in, starpu_data_handle_t out)
- struct starpu_task ∗ starpufft_start_handle (starpufft_plan p, starpu_data_handle_t in, starpu_data_handle_t out)
- void starpufft_cleanup (starpufft_plan p)
- void starpufft_destroy_plan (starpufft_plan p)
- void **starpufft_startstats** (void)
- void **starpufft_stopstats** (void)
- void **starpufft_showstats** (FILE ∗out)
- starpufftf_plan **starpufftf_plan_dft_1d** (int n, int sign, unsigned flags)
- starpufftf_plan **starpufftf_plan_dft_2d** (int n, int m, int sign, unsigned flags)
- starpufftf_plan **starpufftf_plan_dft_3d** (int n, int m, int p, int sign, unsigned flags)
- starpufftf_plan **starpufftf_plan_dft_r2c_1d** (int n, unsigned flags)
- starpufftf_plan **starpufftf_plan_dft_c2r_1d** (int n, unsigned flags)
- void ∗ **starpufftf_malloc** (size_t n)

- void **starpufftf_free** (void ∗p, size_t dim)
- int **starpufftf_execute** (starpufftf_plan p, void ∗in, void ∗out)
- struct starpu_task ∗ **starpufftf_start** (starpufftf_plan p, void ∗in, void ∗out)
- int **starpufftf_execute_handle** (starpufftf_plan p, starpu_data_handle_t in, starpu_data_handle_t out)
- struct starpu_task ∗ **starpufftf_start_handle** (starpufftf_plan p, starpu_data_handle_t in, starpu_data_handle_t out)
- void **starpufftf_cleanup** (starpufftf_plan p)
- void **starpufftf_destroy_plan** (starpufftf_plan p)
- void **starpufftf_startstats** (void)
- void **starpufftf_stopstats** (void)
- void **starpufftf_showstats** (FILE ∗out)
- starpufftl_plan **starpufftl_plan_dft_1d** (int n, int sign, unsigned flags)
- starpufftl_plan **starpufftl_plan_dft_2d** (int n, int m, int sign, unsigned flags)
- starpufftl_plan **starpufftl_plan_dft_3d** (int n, int m, int p, int sign, unsigned flags)
- starpufftl_plan **starpufftl_plan_dft_r2c_1d** (int n, unsigned flags)
- starpufftl_plan **starpufftl_plan_dft_c2r_1d** (int n, unsigned flags)
- void ∗ **starpufftl_malloc** (size_t n)
- void **starpufftl_free** (void ∗p, size_t dim)
- int **starpufftl_execute** (starpufftl_plan p, void ∗in, void ∗out)
- struct starpu_task ∗ **starpufftl_start** (starpufftl_plan p, void ∗in, void ∗out)
- int **starpufftl_execute_handle** (starpufftl_plan p, starpu_data_handle_t in, starpu_data_handle_t out)
- struct starpu_task ∗ **starpufftl_start_handle** (starpufftl_plan p, starpu_data_handle_t in, starpu_data_handle_t out)
- void **starpufftl_cleanup** (starpufftl_plan p)
- void **starpufftl_destroy_plan** (starpufftl_plan p)
- void **starpufftl_startstats** (void)
- void **starpufftl_stopstats** (void)
- void **starpufftl_showstats** (FILE ∗out)

Variables

- int **starpufft_last_plan_number**

59.54 sc_hypervisor.h File Reference

```
#include <starpu.h>
#include <starpu_sched_ctx_hypervisor.h>
#include <sc_hypervisor_config.h>
#include <sc_hypervisor_monitoring.h>
#include <math.h>
```

Data Structures

- struct sc_hypervisor_policy

Functions

- void ∗ sc_hypervisor_init (struct sc_hypervisor_policy ∗policy)
- void sc_hypervisor_shutdown (void)
- void sc_hypervisor_register_ctx (unsigned sched_ctx, double total_flops)
- void sc_hypervisor_unregister_ctx (unsigned sched_ctx)
- void sc_hypervisor_post_resize_request (unsigned sched_ctx, int task_tag)
- void sc_hypervisor_resize_ctxs (unsigned ∗sched_ctxs, int nsched_ctxs, int ∗workers, int nworkers)
- void sc_hypervisor_stop_resize (unsigned sched_ctx)

- void sc_hypervisor_start_resize (unsigned sched_ctx)
- const char ∗ sc_hypervisor_get_policy (void)
- void sc_hypervisor_add_workers_to_sched_ctx (int ∗workers_to_add, unsigned nworkers_to_add, unsigned sched_ctx)
- void sc_hypervisor_remove_workers_from_sched_ctx (int ∗workers_to_remove, unsigned nworkers_to_remove, unsigned sched_ctx, unsigned now)
- void sc_hypervisor_move_workers (unsigned sender_sched_ctx, unsigned receiver_sched_ctx, int ∗workers_to_move, unsigned nworkers_to_move, unsigned now)
- void sc_hypervisor_size_ctxs (unsigned ∗sched_ctxs, int nsched_ctxs, int ∗workers, int nworkers)
- unsigned sc_hypervisor_get_size_req (unsigned ∗∗sched_ctxs, int ∗nsched_ctxs, int ∗∗workers, int ∗nworkers)
- void sc_hypervisor_save_size_req (unsigned ∗sched_ctxs, int nsched_ctxs, int ∗workers, int nworkers)
- void sc_hypervisor_free_size_req (void)
- unsigned sc_hypervisor_can_resize (unsigned sched_ctx)
- void sc_hypervisor_set_type_of_task (struct starpu_codelet ∗cl, unsigned sched_ctx, uint32_t footprint, size_t data_size)
- void sc_hypervisor_update_diff_total_flops (unsigned sched_ctx, double diff_total_flops)
- void sc_hypervisor_update_diff_elapsed_flops (unsigned sched_ctx, double diff_task_flops)
- void sc_hypervisor_update_resize_interval (unsigned ∗sched_ctxs, int nsched_ctxs, int max_nworkers)
- void sc_hypervisor_get_ctxs_on_level (unsigned ∗∗sched_ctxs, int ∗nsched_ctxs, unsigned hierarchy_level, unsigned father_sched_ctx_id)
- unsigned sc_hypervisor_get_nhierarchy_levels (void)
- void sc_hypervisor_get_leaves (unsigned ∗sched_ctxs, int nsched_ctxs, unsigned ∗leaves, int ∗nleaves)
- double sc_hypervisor_get_nready_flops_of_all_sons_of_sched_ctx (unsigned sched_ctx)
- void **sc_hypervisor_print_overhead** (void)
- void **sc_hypervisor_init_worker** (int workerid, unsigned sched_ctx)

Variables

- starpu_pthread_mutex_t act_hypervisor_mutex

59.55 sc_hypervisor_config.h File Reference

`#include <sc_hypervisor.h>`

Data Structures

- struct sc_hypervisor_policy_config

- #define SC_HYPERVISOR_MAX_IDLE
- #define **SC_HYPERVISOR_MIN_WORKING**
- #define SC_HYPERVISOR_PRIORITY
- #define SC_HYPERVISOR_MIN_WORKERS
- #define SC_HYPERVISOR_MAX_WORKERS
- #define SC_HYPERVISOR_GRANULARITY
- #define SC_HYPERVISOR_FIXED_WORKERS
- #define SC_HYPERVISOR_MIN_TASKS
- #define SC_HYPERVISOR_NEW_WORKERS_MAX_IDLE
- #define SC_HYPERVISOR_TIME_TO_APPLY
- #define SC_HYPERVISOR_NULL
- #define SC_HYPERVISOR_ISPEED_W_SAMPLE
- #define SC_HYPERVISOR_ISPEED_CTX_SAMPLE
- #define **SC_HYPERVISOR_TIME_SAMPLE**
- #define **MAX_IDLE_TIME**

- #define **MIN_WORKING_TIME**
- void sc_hypervisor_set_config (unsigned sched_ctx, void ∗config)
- struct sc_hypervisor_policy_config ∗ sc_hypervisor_get_config (unsigned sched_ctx)
- void sc_hypervisor_ctl (unsigned sched_ctx,...)

59.55.1 Data Structure Documentation

59.55.1.1 struct sc_hypervisor_policy_config

Methods that implement a hypervisor resizing policy.

Data Fields

int	min_nworkers	Indicate the minimum number of workers needed by the context
int	max_nworkers	Indicate the maximum number of workers needed by the context
int	granularity	Indicate the workers granularity of the context
int	priority[STARPU_NMAXWORKERS]	Indicate the priority of each worker to stay in the context the smaller the priority the faster it will be moved to another context
double	max_idle[STARPU_NMAXWORKERS]	Indicate the maximum idle time accepted before a resize is triggered above this limit the priority of the worker is reduced
double	min_working[STARPU_NMAXWORKERS]	Indicate that underneath this limit the priority of the worker is reduced
int	fixed_workers[STARPU_NMAXWORKERS]	Indicate which workers can be moved and which ones are fixed
double	new_workers_max_idle	Indicate the maximum idle time accepted before a resize is triggered for the workers that just arrived in the new context
double	ispeed_w_sample[STARPU_NMAXWORKERS]	Indicate the sample used to compute the instant speed per worker
double	ispeed_ctx_sample	Indicate the sample used to compute the instant speed per ctxs
double	time_sample	Indicate the sample used to compute the instant speed per ctx (in seconds)

59.56 sc_hypervisor_lp.h File Reference

```
#include <sc_hypervisor.h>
#include <starpu_config.h>
#include <glpk.h>
```

Functions

- double sc_hypervisor_lp_get_nworkers_per_ctx (int nsched_ctxs, int ntypes_of_workers, double res[nsched_ctxs][ntypes_of_workers], int total_nw[ntypes_of_workers], struct types_of_workers ∗tw, unsigned ∗in_sched_ctxs)
- double sc_hypervisor_lp_get_tmax (int nw, int ∗workers)
- void sc_hypervisor_lp_round_double_to_int (int ns, int nw, double res[ns][nw], int res_rounded[ns][nw])
- void sc_hypervisor_lp_redistribute_resources_in_ctxs (int ns, int nw, int res_rounded[ns][nw], double res[ns][nw], unsigned ∗sched_ctxs, struct types_of_workers ∗tw)

- void sc_hypervisor_lp_distribute_resources_in_ctxs (unsigned ∗sched_ctxs, int ns, int nw, int res_rounded[ns][nw], double res[ns][nw], int ∗workers, int nworkers, struct types_of_workers ∗tw)
- void sc_hypervisor_lp_distribute_floating_no_resources_in_ctxs (unsigned ∗sched_ctxs, int ns, int nw, double res[ns][nw], int ∗workers, int nworkers, struct types_of_workers ∗tw)
- void sc_hypervisor_lp_place_resources_in_ctx (int ns, int nw, double w_in_s[ns][nw], unsigned ∗sched_ctxs, int ∗workers, unsigned do_size, struct types_of_workers ∗tw)
- void sc_hypervisor_lp_share_remaining_resources (int ns, unsigned ∗sched_ctxs, int nworkers, int ∗workers)
- double sc_hypervisor_lp_find_tmax (double t1, double t2)
- unsigned sc_hypervisor_lp_execute_dichotomy (int ns, int nw, double w_in_s[ns][nw], unsigned solve_lp_integer, void ∗specific_data, double tmin, double tmax, double smallest_tmax, double(∗lp_estimated_distrib_func)(int lns, int lnw, double ldraft_w_in_s[ns][nw], unsigned lis_integer, double ltmax, void ∗lspecifc_data))
- double sc_hypervisor_lp_simulate_distrib_flops (int nsched_ctxs, int ntypes_of_workers, double speed[nsched_ctxs][ntypes_of_workers], double flops[nsched_ctxs], double res[nsched_ctxs][ntypes_of_workers], int total_nw[ntypes_of_workers], unsigned sched_ctxs[nsched_ctxs], double vmax)
- double sc_hypervisor_lp_simulate_distrib_tasks (int ns, int nw, int nt, double w_in_s[ns][nw], double tasks[nw][nt], double times[nw][nt], unsigned is_integer, double tmax, unsigned ∗in_sched_ctxs, struct sc_hypervisor_policy_task_pool ∗tmp_task_pools)
- double sc_hypervisor_lp_simulate_distrib_flops_on_sample (int ns, int nw, double final_w_in_s[ns][nw], unsigned is_integer, double tmax, double ∗∗speed, double flops[ns], double ∗∗final_flops_on_w)

59.57 sc_hypervisor_monitoring.h File Reference

`#include <sc_hypervisor.h>`

Data Structures

- struct sc_hypervisor_resize_ack
- struct sc_hypervisor_wrapper

Functions

- struct sc_hypervisor_wrapper ∗ sc_hypervisor_get_wrapper (unsigned sched_ctx)
- unsigned ∗ sc_hypervisor_get_sched_ctxs (void)
- int sc_hypervisor_get_nsched_ctxs (void)
- int sc_hypervisor_get_nworkers_ctx (unsigned sched_ctx, enum starpu_worker_archtype arch)
- double sc_hypervisor_get_elapsed_flops_per_sched_ctx (struct sc_hypervisor_wrapper ∗sc_w)
- double sc_hypervisor_get_total_elapsed_flops_per_sched_ctx (struct sc_hypervisor_wrapper ∗sc_w)
- double sc_hypervisorsc_hypervisor_get_speed_per_worker_type (struct sc_hypervisor_wrapper ∗sc_w, enum starpu_worker_archtype arch)
- double sc_hypervisor_get_speed (struct sc_hypervisor_wrapper ∗sc_w, enum starpu_worker_archtype arch)

59.57.1 Data Structure Documentation

59.57.1.1 struct sc_hypervisor_wrapper

Wrapper of the contexts available in StarPU which contains all information about a context obtained by incrementing the performance counters. it is attached to a sched_ctx storing monitoring information

Data Fields

unsigned	sched_ctx	the monitored context
struct sc_hypervisor_policy_config ∗	config	The corresponding resize configuration

Data Fields

double	start_time_w[STARPU_NMAXWORKERS]	The start time of the resizing sample of the workers of this context
double	current_idle_time[STARPU_NMAXWORKERS]	The idle time counter of each worker of the context
double	idle_time[STARPU_NMAXWORKERS]	The time the workers were idle from the last resize
double	idle_start_time[STARPU_NMAXWORKERS]	The moment when the workers started being idle
double	exec_time[STARPU_NMAXWORKERS]	Time during which the worker executed tasks
double	exec_start_time[STARPU_NMAXWORKERS]	Time when the worker started executing a task
int	worker_to_be_removed[STARPU_NMAXWORKERS]	List of workers that will leave the context (lazy resizing process)
int	pushed_tasks[STARPU_NMAXWORKERS]	Number of tasks pushed on each worker in this context
int	poped_tasks[STARPU_NMAXWORKERS]	Number of tasks poped from each worker in this context
double	total_flops	The total number of flops to execute by the context
double	total_elapsed_flops[STARPU_NMAXWORKERS]	The number of flops executed by each workers of the context
double	elapsed_flops[STARPU_NMAXWORKERS]	number of flops executed since last resizing
size_t	elapsed_data[STARPU_NMAXWORKERS]	Quantity of data (in bytes) used to execute tasks on each worker in this context
int	elapsed_tasks[STARPU_NMAXWORKERS]	Number of tasks executed on each worker in this context
double	ref_speed[2]	the average speed of the type of workers when they belonged to this context 0 - cuda 1 - cpu
double	submitted_flops	Number of flops submitted to this context
double	remaining_flops	Number of flops that still have to be executed by the workers in this context
double	start_time	Start time of the resizing sample of this context
double	real_start_time	First time a task was pushed to this context
double	hyp_react_start_time	Start time for sample in which the hypervisor is not allowed to react bc too expensive
struct sc_hypervisor_resize_ack	resize_ack	Structure confirming the last resize finished and a new one can be done. Workers do not leave the current context until the receiver context does not ack the receive of these workers
starpu_pthread_mutex_t	mutex	Mutex needed to synchronize the acknowledgment of the workers into the receiver context

Data Fields

unsigned	total_flops_available	Boolean indicating if the hypervisor can use the flops corresponding to the entire execution of the context
unsigned	to_be_sized	boolean indicating that a context is being sized
unsigned	compute_idle[STARPU_NMAXWORKERS]	Boolean indicating if we add the idle of this worker to the idle of the context
unsigned	compute_partial_idle[STARPU_NMAXWORKERS]	Boolean indicating if we add the entiere idle of this worker to the idle of the context or just half
unsigned	consider_max	consider the max in the lp

59.58 sc_hypervisor_policy.h File Reference

`#include <sc_hypervisor.h>`

Data Structures

- struct types_of_workers
- struct sc_hypervisor_policy_task_pool

Macros

- #define **HYPERVISOR_REDIM_SAMPLE**
- #define **HYPERVISOR_START_REDIM_SAMPLE**
- #define **SC_NOTHING**
- #define **SC_IDLE**
- #define **SC_SPEED**

Functions

- void sc_hypervisor_policy_add_task_to_pool (struct starpu_codelet ∗cl, unsigned sched_ctx, uint32_t footprint, struct sc_hypervisor_policy_task_pool ∗∗task_pools, size_t data_size)
- void sc_hypervisor_policy_remove_task_from_pool (struct starpu_task ∗task, uint32_t footprint, struct sc_hypervisor_policy_task_pool ∗∗task_pools)
- struct sc_hypervisor_policy_task_pool ∗ sc_hypervisor_policy_clone_task_pool (struct sc_hypervisor_policy_task_pool ∗tp)
- void sc_hypervisor_get_tasks_times (int nw, int nt, double times[nw][nt], int ∗workers, unsigned size_ctxs, struct sc_hypervisor_policy_task_pool ∗task_pools)
- unsigned sc_hypervisor_find_lowest_prio_sched_ctx (unsigned req_sched_ctx, int nworkers_to_move)
- int ∗ sc_hypervisor_get_idlest_workers (unsigned sched_ctx, int ∗nworkers, enum starpu_worker_archtype arch)
- int ∗ sc_hypervisor_get_idlest_workers_in_list (int ∗start, int ∗workers, int nall_workers, int ∗nworkers, enum starpu_worker_archtype arch)
- int sc_hypervisor_get_movable_nworkers (struct sc_hypervisor_policy_config ∗config, unsigned sched_ctx, enum starpu_worker_archtype arch)
- int sc_hypervisor_compute_nworkers_to_move (unsigned req_sched_ctx)
- unsigned sc_hypervisor_policy_resize (unsigned sender_sched_ctx, unsigned receiver_sched_ctx, unsigned force_resize, unsigned now)
- unsigned sc_hypervisor_policy_resize_to_unknown_receiver (unsigned sender_sched_ctx, unsigned now)

- double sc_hypervisor_get_ctx_speed (struct sc_hypervisor_wrapper ∗sc_w)
- double sc_hypervisor_get_slowest_ctx_exec_time (void)
- double sc_hypervisor_get_fastest_ctx_exec_time (void)
- double sc_hypervisor_get_speed_per_worker (struct sc_hypervisor_wrapper ∗sc_w, unsigned worker)
- double sc_hypervisor_get_speed_per_worker_type (struct sc_hypervisor_wrapper ∗sc_w, enum starpu_worker_archtype arch)
- double sc_hypervisor_get_ref_speed_per_worker_type (struct sc_hypervisor_wrapper ∗sc_w, enum starpu_worker_archtype arch)
- double sc_hypervisor_get_avg_speed (enum starpu_worker_archtype arch)
- void sc_hypervisor_check_if_consider_max (struct types_of_workers ∗tw)
- void sc_hypervisor_group_workers_by_type (struct types_of_workers ∗tw, int ∗total_nw)
- enum starpu_worker_archtype sc_hypervisor_get_arch_for_index (unsigned w, struct types_of_workers ∗tw)
- unsigned sc_hypervisor_get_index_for_arch (enum starpu_worker_archtype arch, struct types_of_workers ∗tw)
- unsigned sc_hypervisor_criteria_fulfilled (unsigned sched_ctx, int worker)
- unsigned sc_hypervisor_check_idle (unsigned sched_ctx, int worker)
- unsigned sc_hypervisor_check_speed_gap_btw_ctxs (unsigned ∗sched_ctxs, int nsched_ctxs, int ∗workers, int nworkers)
- unsigned sc_hypervisor_check_speed_gap_btw_ctxs_on_level (int level, int ∗workers_in, int nworkers_in, unsigned father_sched_ctx_id, unsigned ∗∗sched_ctxs, int ∗nsched_ctxs)
- unsigned sc_hypervisor_get_resize_criteria (void)
- struct types_of_workers ∗ sc_hypervisor_get_types_of_workers (int ∗workers, unsigned nworkers)

59.59 starpurm.h File Reference

```
#include <hwloc.h>
#include <starpurm_config.h>
```

Typedefs

- typedef int **starpurm_drs_ret_t**
- typedef void ∗ **starpurm_drs_desc_t**
- typedef void ∗ **starpurm_drs_cbs_t**
- typedef void(∗ **starpurm_drs_cb_t**) (void ∗)
- typedef void ∗ **starpurm_block_cond_t**
- typedef int(∗ **starpurm_polling_t**) (void ∗)

Enumerations

- enum e_starpurm_drs_ret { starpurm_DRS_SUCCESS , starpurm_DRS_DISABLD , starpurm_DRS_PERM , starpurm_DRS_EINVAL }

Functions

Initialisation

- void starpurm_initialize_with_cpuset (hwloc_cpuset_t initially_owned_cpuset)
- void starpurm_initialize (void)
- void starpurm_shutdown (void)

Spawn

- void starpurm_spawn_kernel_on_cpus (void ∗data, void(∗f)(void ∗), void ∗args, hwloc_cpuset_t cpuset)
- void starpurm_spawn_kernel_on_cpus_callback (void ∗data, void(∗f)(void ∗), void ∗args, hwloc_cpuset_t cpuset, void(∗cb_f)(void ∗), void ∗cb_args)

- void **starpurm_spawn_kernel_callback** (void ∗data, void(∗f)(void ∗), void ∗args, void(∗cb_f)(void ∗), void ∗cb_args)

DynamicResourceSharing

- starpurm_drs_ret_t starpurm_set_drs_enable (starpurm_drs_desc_t ∗spd)
- starpurm_drs_ret_t starpurm_set_drs_disable (starpurm_drs_desc_t ∗spd)
- int starpurm_drs_enabled_p (void)
- starpurm_drs_ret_t starpurm_set_max_parallelism (starpurm_drs_desc_t ∗spd, int max)
- starpurm_drs_ret_t starpurm_assign_cpu_to_starpu (starpurm_drs_desc_t ∗spd, int cpuid)
- starpurm_drs_ret_t starpurm_assign_cpus_to_starpu (starpurm_drs_desc_t ∗spd, int ncpus)
- starpurm_drs_ret_t starpurm_assign_cpu_mask_to_starpu (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_t mask)
- starpurm_drs_ret_t starpurm_assign_all_cpus_to_starpu (starpurm_drs_desc_t ∗spd)
- starpurm_drs_ret_t starpurm_withdraw_cpu_from_starpu (starpurm_drs_desc_t ∗spd, int cpuid)
- starpurm_drs_ret_t starpurm_withdraw_cpus_from_starpu (starpurm_drs_desc_t ∗spd, int ncpus)
- starpurm_drs_ret_t starpurm_withdraw_cpu_mask_from_starpu (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_t mask)
- starpurm_drs_ret_t starpurm_withdraw_all_cpus_from_starpu (starpurm_drs_desc_t ∗spd)
- starpurm_drs_ret_t starpurm_lend (starpurm_drs_desc_t ∗spd)
- starpurm_drs_ret_t starpurm_lend_cpu (starpurm_drs_desc_t ∗spd, int cpuid)
- starpurm_drs_ret_t starpurm_lend_cpus (starpurm_drs_desc_t ∗spd, int ncpus)
- starpurm_drs_ret_t starpurm_lend_cpu_mask (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_t mask)
- starpurm_drs_ret_t starpurm_reclaim (starpurm_drs_desc_t ∗spd)
- starpurm_drs_ret_t starpurm_reclaim_cpu (starpurm_drs_desc_t ∗spd, int cpuid)
- starpurm_drs_ret_t starpurm_reclaim_cpus (starpurm_drs_desc_t ∗spd, int ncpus)
- starpurm_drs_ret_t starpurm_reclaim_cpu_mask (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_t mask)
- starpurm_drs_ret_t starpurm_acquire (starpurm_drs_desc_t ∗spd)
- starpurm_drs_ret_t starpurm_acquire_cpu (starpurm_drs_desc_t ∗spd, int cpuid)
- starpurm_drs_ret_t starpurm_acquire_cpus (starpurm_drs_desc_t ∗spd, int ncpus)
- starpurm_drs_ret_t starpurm_acquire_cpu_mask (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_t mask)
- starpurm_drs_ret_t starpurm_return_all (starpurm_drs_desc_t ∗spd)
- starpurm_drs_ret_t starpurm_return_cpu (starpurm_drs_desc_t ∗spd, int cpuid)

Devices

- int starpurm_get_device_type_id (const char ∗type_str)
- const char ∗ starpurm_get_device_type_name (int type_id)
- int starpurm_get_nb_devices_by_type (int type_id)
- int starpurm_get_device_id (int type_id, int device_rank)
- starpurm_drs_ret_t starpurm_assign_device_to_starpu (starpurm_drs_desc_t ∗spd, int type_id, int unit_rank)
- starpurm_drs_ret_t starpurm_assign_devices_to_starpu (starpurm_drs_desc_t ∗spd, int type_id, int ndevices)
- starpurm_drs_ret_t starpurm_assign_device_mask_to_starpu (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_t mask)
- starpurm_drs_ret_t starpurm_assign_all_devices_to_starpu (starpurm_drs_desc_t ∗spd, int type_id)
- starpurm_drs_ret_t starpurm_withdraw_device_from_starpu (starpurm_drs_desc_t ∗spd, int type_id, int unit_rank)
- starpurm_drs_ret_t starpurm_withdraw_devices_from_starpu (starpurm_drs_desc_t ∗spd, int type_id, int ndevices)
- starpurm_drs_ret_t starpurm_withdraw_device_mask_from_starpu (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_t mask)
- starpurm_drs_ret_t starpurm_withdraw_all_devices_from_starpu (starpurm_drs_desc_t ∗spd, int type_id)
- starpurm_drs_ret_t starpurm_lend_device (starpurm_drs_desc_t ∗spd, int type_id, int unit_rank)
- starpurm_drs_ret_t starpurm_lend_devices (starpurm_drs_desc_t ∗spd, int type_id, int ndevices)
- starpurm_drs_ret_t starpurm_lend_device_mask (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_t mask)
- starpurm_drs_ret_t starpurm_lend_all_devices (starpurm_drs_desc_t ∗spd, int type_id)
- starpurm_drs_ret_t starpurm_reclaim_device (starpurm_drs_desc_t ∗spd, int type_id, int unit_rank)
- starpurm_drs_ret_t starpurm_reclaim_devices (starpurm_drs_desc_t ∗spd, int type_id, int ndevices)

- starpurm_drs_ret_t starpurm_reclaim_device_mask (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_↩ t mask)
- starpurm_drs_ret_t starpurm_reclaim_all_devices (starpurm_drs_desc_t ∗spd, int type_id)
- starpurm_drs_ret_t starpurm_acquire_device (starpurm_drs_desc_t ∗spd, int type_id, int unit_rank)
- starpurm_drs_ret_t starpurm_acquire_devices (starpurm_drs_desc_t ∗spd, int type_id, int ndevices)
- starpurm_drs_ret_t starpurm_acquire_device_mask (starpurm_drs_desc_t ∗spd, const hwloc_cpuset_↩ t mask)
- starpurm_drs_ret_t starpurm_acquire_all_devices (starpurm_drs_desc_t ∗spd, int type_id)
- starpurm_drs_ret_t starpurm_return_all_devices (starpurm_drs_desc_t ∗spd, int type_id)
- starpurm_drs_ret_t starpurm_return_device (starpurm_drs_desc_t ∗spd, int type_id, int unit_rank)

CpusetsQueries

- hwloc_cpuset_t starpurm_get_device_worker_cpuset (int type_id, int unit_rank)
- hwloc_cpuset_t starpurm_get_global_cpuset (void)
- hwloc_cpuset_t starpurm_get_selected_cpuset (void)
- hwloc_cpuset_t starpurm_get_all_cpu_workers_cpuset (void)
- hwloc_cpuset_t starpurm_get_all_device_workers_cpuset (void)
- hwloc_cpuset_t starpurm_get_all_device_workers_cpuset_by_type (int typeid)

Chapter 60

Deprecated List

Global STARPU_CLUSTER_AWAKE_WORKERS
Use STARPU_PARALLEL_WORKER_AWAKE_WORKERS

Global STARPU_CLUSTER_CREATE_FUNC
Use STARPU_PARALLEL_WORKER_CREATE_FUNC

Global STARPU_CLUSTER_CREATE_FUNC_ARG
Use STARPU_PARALLEL_WORKER_CREATE_FUNC_ARG

Global STARPU_CLUSTER_KEEP_HOMOGENEOUS
Use STARPU_PARALLEL_WORKER_KEEP_HOMOGENEOUS

Global starpu_cluster_machine (hwloc_obj_type_t cluster_level,...)
Use starpu_parallel_worker_init()

Global STARPU_CLUSTER_MAX_NB
Use STARPU_PARALLEL_WORKER_MAX_NB

Global STARPU_CLUSTER_MIN_NB
Use STARPU_PARALLEL_WORKER_MIN_NB

Global STARPU_CLUSTER_NB
Use STARPU_PARALLEL_WORKER_NB

Global STARPU_CLUSTER_NCORES
Use STARPU_PARALLEL_WORKER_NCORES

Global STARPU_CLUSTER_NEW
Use STARPU_PARALLEL_WORKER_NEW

Global STARPU_CLUSTER_PARTITION_ONE
Use STARPU_PARALLEL_WORKER_PARTITION_ONE

Global STARPU_CLUSTER_POLICY_NAME
Use STARPU_PARALLEL_WORKER_POLICY_NAME

Global STARPU_CLUSTER_POLICY_STRUCT
Use STARPU_PARALLEL_WORKER_POLICY_STRUCT

Global STARPU_CLUSTER_PREFERE_MIN
Use STARPU_PARALLEL_WORKER_PREFERE_MIN

Global starpu_cluster_print (struct starpu_cluster_machine ∗clusters)
Use starpu_parallel_worker_print()

Global STARPU_CLUSTER_TYPE
Use STARPU_PARALLEL_WORKER_TYPE

Global starpu_cluster_types
Use starpu_parallel_worker_types

Global starpu_codelet::cpu_func

Optional field which has been made deprecated. One should use instead the field starpu_codelet::cpu_funcs.

Global starpu_codelet::cuda_func

Optional field which has been made deprecated. One should use instead the starpu_codelet::cuda_funcs field.

Global starpu_codelet::opencl_func

Optional field which has been made deprecated. One should use instead the starpu_codelet::opencl_funcs field.

Global starpu_data_free_pinned_if_possible

Equivalent to starpu_free(). This macro is provided to avoid breaking old codes.

Global starpu_data_interface_ops::handle_to_pointer)(starpu_data_handle_t handle, unsigned node)

Use starpu_data_interface_ops::to_pointer instead. Return the current pointer (if any) for the handle on the given node.

Global starpu_data_malloc_pinned_if_possible

Equivalent to starpu_malloc(). This macro is provided to avoid breaking old codes.

Global starpu_free (void *A)

Free memory which has previously been allocated with starpu_malloc(). This function is deprecated, one should use starpu_free_noflag(). See Data Management Allocation for more details.

Global starpu_mpi_initialize (void)

This function has been made deprecated. One should use instead the function starpu_mpi_init(). This function does not call `MPI_Init()`, it should be called beforehand.

Global starpu_mpi_initialize_extended (int *rank, int *world_size)

This function has been made deprecated. One should use instead the function starpu_mpi_init(). MPI will be initialized by starpumpi by calling `MPI_Init_Thread(argc, argv, MPI_THREAD_SERIALIZED, ...)`.

Global STARPU_MULTIPLE_CPU_IMPLEMENTATIONS

Setting the field starpu_codelet::cpu_func with this macro indicates the codelet will have several implementations. The use of this macro is deprecated. One should always only define the field starpu_codelet::cpu_funcs.

Global STARPU_MULTIPLE_CUDA_IMPLEMENTATIONS

Setting the field starpu_codelet::cuda_func with this macro indicates the codelet will have several implementations. The use of this macro is deprecated. One should always only define the field starpu_codelet::cuda_funcs.

Global STARPU_MULTIPLE_HIP_IMPLEMENTATIONS

Setting the field starpu_codelet::hip_func with this macro indicates the codelet will have several implementations. The use of this macro is deprecated. One should always only define the field starpu_codelet::hip_funcs.

Global STARPU_MULTIPLE_OPENCL_IMPLEMENTATIONS

Setting the field starpu_codelet::opencl_func with this macro indicates the codelet will have several implementations. The use of this macro is deprecated. One should always only define the field starpu_codelet::opencl_funcs.

Global starpu_uncluster_machine (struct starpu_cluster_machine *clusters)

Use starpu_parallel_worker_shutdown()

Index